2

CECOM CENTER
FOR
SOFTWARE ENGINEERING

SOFTWARE METHODOLOGY CATALOG

Second Edition

- Revision

CENTER FOR QUALITY SUPPORT
CSE
CECOM
SOFTWARE ENGINEERING

2

RESEARCH AND DEVELOPMENT TECHNICAL REPORT
C01-091JB-0001-01

SOFTWARE METHODOLOGY CATALOG

Second Edition

Prepared by

Laurel Von Gerichten
Marilyn Ginsberg
Richard Pirchner
Richard Guilfoyle

Teledyne Brown Engineering
151 Industrial Way East
Eatontown, NJ 07724

March 1989

Approved for public release; distribution is unlimited.

Prepared for

CECOM CENTER FOR SOFTWARE ENGINEERING

US ARMY COMMUNICATIONS-ELECTRONICS COMMAND
FORT MONMOUTH, NEW JERSEY 07703-5000

NOTICES

Disclaimer

The citation of trade names and names of manufacturers in this report is not to be construed as official Government indorsement or approval of commercial products or services referenced herein.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION
Unclassified

1b. RESTRICTIVE MARKINGS

2a. SECURITY CLASSIFICATION AUTHORITY

2b. DECLASSIFICATION / DOWNGRADING SCHEDULE

3. DISTRIBUTION / AVAILABILITY OF REPORT
Approved for public release; distribution is unlimited.

4. PERFORMING ORGANIZATION REPORT NUMBER(S)

5. MONITORING ORGANIZATION REPORT NUMBER(S)
C01-091JB-0001-01

6a. NAME OF PERFORMING ORGANIZATION
Teledyne Brown Engineering

6b. OFFICE SYMBOL (If applicable)

7a. NAME OF MONITORING ORGANIZATION
US Army Communications-Electronics Command (CECOM), Center for Software Engineering

6c. ADDRESS (City, State, and ZIP Code)
151 Industrial Way East
Eatontown, NJ 07724

7b. ADDRESS (City, State, and ZIP Code)
ATTN: AMSEL-RD-SE-AST-SE
Fort Monmouth, NJ 07703-5000

8a. NAME OF FUNDING / SPONSORING ORGANIZATION

8b. OFFICE SYMBOL (If applicable)

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
DAAB07-86-D-R001

8c. ADDRESS (City, State, and ZIP Code)

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.
612783	A094	01	01

11. TITLE (Include Security Classification)
SOFTWARE METHODOLOGY CATALOG, SECOND EDITION (U)

12. PERSONAL AUTHOR(S)
Laurel Von Gerichten, Marilyn Ginsberg, Richard Pirchner, Richard Guilfoyle

13a. TYPE OF REPORT
Technical Report

13b. TIME COVERED
FROM Apr 88 TO Mar 89

14. DATE OF REPORT (Year, Month, Day)
1989 March

15. PAGE COUNT
445

16. SUPPLEMENTARY NOTATION
This report is a revision of Report No. C01-901JB-0001, same title, dated December 1988, ADB128594.

17. COSATI CODES

FIELD	GROUP	SUB-GROUP
12	05	

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Software Methodology; Software Engineering; Computer Programming

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This technical report provides a consolidated reference for software methods used over the total spectrum of software development. The primary objective is to provide, for each method included, a brief overview of the method and to give some insight into the underlying assumptions, the software development activities which it supports, and other characteristics associated with its use. A second objective is to provide sources for further information. The second edition of the catalog reports on 73 software methods.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION
Unclassified

22a. NAME OF RESPONSIBLE INDIVIDUAL
Harold L. Tamburro

22b. TELEPHONE (Include Area Code)
(201) 544-2029

22c. OFFICE SYMBOL
AMSEL-RD-SE-AST-SE

Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

TABLE OF CONTENTS

CHAPTER 1: CATALOG OVERVIEW

CHAPTER 2: AN OVERVIEW OF BASIC CONCEPTS

CHAPTER 3: METHODS CURRENTLY IN USE

CHAPTER 4: EMERGING METHODS

This page is intentionally left blank.

CHAPTER 1

CATALOG OVERVIEW

1.1 INTRODUCTION

Purpose

This catalog provides a consolidated reference for methods used over the total spectrum of software development. A primary objective is to provide a brief overview of each method, and to give for each method some insight into its underlying assumptions, the software development activities which it supports, and other characteristics associated with its use. A second objective is to provide sources for further information, including literature references and points of contact. One final objective is to provide a facility for contrasting the various methods relative to selected attributes.

The software process is but one component of the larger goal of building a computer-based system. There are few methods which address the total development process. Although some of the methods described in the catalog deal with total system development, it is the purpose of this catalog to focus upon methods associated with developing the system software. The catalog represents an effort to obtain the most current information available for each method. To this end, surveys were developed, and responses were solicited from developers. In writing the catalog the authors have elected to serve as reporters rather than evaluators. Evaluative statements that appear in the catalog are based upon survey responses.

Intended Audience

The catalog is intended for software engineering professionals involved in either the technical or managerial aspects of software development. In order to present each method in a way that the reader would find informative and predictable, a fixed format for describing methods was chosen. It is assumed that the reader has some background experience with software development projects, as well as some familiarity with methods.

The catalog offers the reader brief descriptions of different methods and their associated characteristics. While the descriptions of methods are not of sufficient depth to function as a tutorial, they should provide enough detail to portray the essentials of the method and of related usage characteristics. The catalog also provides a discussion of some of the main ideas in the software development process. This discussion establishes the basic terminology and concepts used in the catalog, and should be consulted for background information and for an understanding of the authors' perspective.

It is hoped that the software engineering community will find this document useful for gaining insight into particular methods, as well as for solidifying an understanding of some of the principal ideas in the field. For this latter purpose, a bibliography has been provided in an appendix. Thus, the catalog may serve as a reference for ones own work, or for understanding software development methods used by others.

1.2 BASIS OF CATALOG

Background

In November, 1982, in a study commissioned by the Ada Joint Program Office, Peter Freeman and Anthony Wasserman set forth the requirements for a software development methodology. This study [Free82], commonly known as Methodman, related software process issues to the Ada Programming Support Environment (APSE) and presented an evaluation of a set of methods. Included in the report were a summary of questionnaire responses and a study plan for evaluating software design methods for use with Ada.

A follow-up study to Methodman was performed by the Institute for Defense Analyses under the auspices of the Software Technology for Adaptable and Reliable Systems (STARS) Joint Program Office [Conv85], [McDo85]. Volume II of the report [McDo85] has become known as Methodman II, and summarizes the work of the STARS Methodology Coordination Team. The report addressed issues related to the total software development process. The report also suggested an approach to classifying, evaluating, and selecting methods.

The two studies above provided much of the initial background for the current catalog. The issues discussed in those reports were reviewed and contrasted with reports of other related research in software engineering. Among the latter was a report published by the Department of Industry, London, in September, 1981 [DInd81]. This study correlated features of the Ada language with the system development process and described a number of well-known methods. In addition, the study provided an informative discussion of software process issues and analyzed the concepts of encapsulation, concurrency, and formalized approaches to software development.

Also reviewed was a report which detailed the problems of NASA's MSOCC [RoyD84]. The approach taken in this study was to identify the "...many influences within the workplace that affect the productivity of software developers..." [Free82]. The study reported on problems in software development as seen from the developer's viewpoint, current trends in methods and tools that might alleviate the problems, and some other projected solutions.

Ideas emanating from various conferences and workshops have further influenced the perspective of this catalog. The International Workshop on the Software Process and Software Environments held in March, 1985, provided definitions and discussions that have aided in reconciling terms and concepts that have evolved in recent times. The presentations and proceedings of the Ninth International Conference on Software Engineering [ICSE87] held in April, 1987, provided a further update to the issues. Additionally, the emergence of a field known as Computer Aided Software Engineering (CASE) has brought about an emphasis on the connection between environments and methods, and has affected the way automated tools have been regarded in the catalog.

Finally, the authors have had an opportunity :o refine the approach to this second edition of the catalog, based on insight gained from their creation of the first edition [Maha87]. The principal difference between the two editions is the approach towards gathering information for the catalog.

Scope of the Catalog

In developing the scope of the catalog, there was a need to clarify the definitions of method, approach, tool, and environment. Definitions given in the Conference Proceedings of the 1985 International Workshop on the Software Process and Software Environments [IWSP85] were used to derive a set of working definitions that follow. In the discussion which follows, the word scheme has been used to mean "a way of performing a set of activities."

- Method: A definite, established, logical, or systematic plan. The steps and purposes have been thought out beforehand in detail. A scheme is a method when it guides the user to a predictable result given an appropriate set of starting conditions.

- Approach: A way of beginning or managing an effort; a way of analyzing, planning or directing a project; a way of conducting operations. A scheme is an approach when it suggests ways to identify goals initially and/or suggests, at an abstract level, ways to proceed toward goals.

- Tool: Anything used to do specialized work or to obtain a specific result; there is a unity of purpose. A scheme is a tool when it is essentially automatic. The user supplies input data or changes, but the tool produces the associated work product.

- Environment: An integrated collection of tools supporting an approach. The components of an environment are designed to reduce the effort required to carry out the software development process whether they are used individually or in combination.

The use of the term method in this catalog needs further explanation. Both tools and approaches exist over wide spectra. Within these spectra, making a clear distinction between a sophisticated tool and a method on the one hand, or between a method and a prescriptive approach on the other, is difficult. Thus, the authors' use of the term method in this catalog has been widened to include schema that could be termed tools or approaches.

By way of examples, consider th‿ following. An integrated set of tools with a prescriptive user's manual might be offered as a software development environment. In such a case, an implied method or approach exists by virtue of the fact that there is only one way to use the set of tools in order to arrive at a final software product. On the other hand consider a scheme which prescribes the order in which activities occur and how these activities are to be managed, but is not limited to one specific way of accomplishing individual activities. This scheme might be offered as an approach. Nevertheless, the instantiation of such a scheme implies a definite systematic plan for developing software. Accordingly, this catalog includes those schema that have been judged to satisfy the essence of the definition for method given above: a definite, established, logical, or systematic plan.

The term methodology occurs often in the context of software development. It was observed that the meaning assigned to the term varied widely throughout the software community. Furthermore, recent practice within the software engineering field has encouraged the use of the term methodology only in reference to the study of methods. Accordingly, the meaning of the term in this catalog is restricted to the study of methods.

Survey Efforts and Results

For a method to be included in the catalog, it was decided that information should be provided by the developer or vendor of the method. Some important methods may have been omitted due to a lack of response from developers who were contacted, or due to inadequate contact information with respect to other developers.

For the first edition, an initial set of candidate methods was compiled from prior methodology surveys, conference proceedings, technical journals, bibliographies published by professional organizations, and responses to a bulletin board notice posted on various nation-wide electronic networks. In some cases, an initial contact letter was used to further clarify the suitability of these candidates. The candidate methods for the second edition were identified based on the efforts of the first edition as well as on updated information. In addition, the authors made a decision to de-emphasize inclusioa of methods which were considered to be single-purpose and oriented towards data-processing systems. Emphasis was placed on obtaining multi-purpose methods as well as methods oriented towards developing real-time systems.

The strategy for obtaining information for the first edition of the catalog differed significantly from the strategy used for the second edition. In the first edition, the survey of method developers was designed such that the responses from this survey could be "verified" by a method-user survey vehicle. This approach was taken in the light of criticisms of previous studies which failed to include the perspectives of users of methods. The questions for both the developer and user surveys for the first edition were, in the main, formulated for multiple-choice response, in order to simplify data analysis and to provide the opportunity to contrast methods.

In the initial phases of the efforts to update the catalog, an analysis was made of the results of the first edition and the inherent data-gathering approach. Based on this analysis, an extensive revision was made of the questionnaire, with emphasis placed on soliciting more specific information on various method aspects. Additionally, an extensive analysis was made of the attempt to gather data from method users. The authors concluded that gathering meaningful data about individual methods from users requires information about the respondents themselves, and would involve significantly more resources than were available in the project. Accordingly, it was decided to use only developer/vendor information as a basis for the second edition of the catalog. Two survey vehicles were developed: a survey for methods currently in use in the software engineering community, and a survey for methods which have not yet become available for general use.

The data gathering results for the second edition were as follows. 143 initial letters were distributed in order to determine the appropriate type of survey to send; from responses to this initial mailing and from sources identifying known methods, approximately 137 developer survey questionnaires were distributed. Completed questionnaires were received for 63 (45%) methods. This total represents the combined response for both types of survey vehicles.

In summary, candidate methods were identified by researching software development literature and recent conference proceedings, and by an initial contact letter sent to professionals in the field. The information used to describe each method was obtained primarily from responses to the developer questionnaire; technical literature provided additional information. The results of the survey also provided the basis for comparing methods to each other.

1.3 CONTENT OF CATALOG

Determination of Descriptive Characteristics

The particular set of characteristics chosen to describe methods was determined in the following manner. An initial set was identified based upon terms used primarily in the two Methodman studies, with some further input from other methodology research sources. This set, based upon the considerable previous efforts of others, provided a foundation for further analysis and refinement.

During the next step in the selection process, the authors individually scored each characteristic for identifiability and importance. The word "identifiability" is used to mean the capability of establishing that a given method has the given characteristic. A consensus rating was then determined. It was evident in this process that various studies were not consistent in the meaning assigned to such characteristics. Additionally, for some characteristics of high importance it was difficult to establish their identifiability. This difficulty was due to one or more factors. In the first case, the characteristic may be an abstraction representing a collection of other characteristics which themselves are more identifiable. Secondly, the characteristic may represent a concept that has come into being before any well-defined sub-characteristics have been associated with it [Abbo83]. Finally, the determination of whether a given method possesses a certain characteristic may require some form of experimentation.

As a result of the above process, the characteristics selected for describing methods in this catalog were:

- **Activities covered by the method,**
- **Extent of usage,**
- **Appropriate application areas,**
- **Ability to incorporate requirements of the target system,**
- **Support of communication during development,**
- **Client involvement,**
- **Support of changeability,**
- **Support of project management,**
- **Automated facilities supporting the method,**
- **Available training in the use of the method,** and
- **Acquisition factors.**

Additionally, some information has been provided about characteristics associated with the resulting software, including maintainability, portability, testability, reliability, and reusability.

It may be observed that these characteristics are similar to those derived in previous studies. Where this analysis does differ from these previous studies is in the emphasis placed on the method itself as opposed to the software resulting from use of the method. When characteristics that apply more to the qualities of the resulting software than to the method are discussed, such as portability or maintainability, the focus is to describe the intent of the method to impart these qualities rather than to evaluate the effectiveness of that intent. Thus, some important software qualities have been de-emphasized in the catalog due to the lack of measurable criteria. What has been provided is a framework for reporting information on a wide range of methods.

Organization of the Catalog

Chapter 1, "Catalog Overview", discusses the procedures that were followed in arriving at the content of this catalog, as well as the organization of the information in this document.

Chapter 2, "An Overview of Basic Concepts", provides background information on software process concepts. Examples of software process models are given, followed by a discussion of approaches to software development. Additionally, the relationships between methods and approaches, as well as between tools and environments, are explored.

Chapter 3, "Methods Currently in Use", presents descriptions of methods which are used to develop software systems, and for which a developer survey was completed.

Chapter 4, "Emerging Methods", presents brief descriptions of additional methods for which a developer survey was returned but which are classified as in development or exploratory.

Chapter 5, "Other Methods Currently in Use", summarizes information on several methods which were included in the first edition but for which more current information could not be obtained.

Chapter 6, "Summaries of Responses to Selected Survey Questions", contains the authors' syntheses of responses to selected developer survey questions. Additionally, summaries of these responses are presented in tabular form.

Appendix A, "Bibliography", is divided into a general reference section and a method-specific section. Individual items have been included either because they were referenced in the catalog or because they provide an additional source of information related to the software process.

Appendix B, "Survey of Software Method Developers", provides a copy of the questionnaire used to compile information about methods currently in use.

Appendix C, "Survey of Emerging Methods", provides a copy of the questionnaire used to compile information about methods not yet available for general use.

Format for Describing Methods

The information which is used to describe an individual method is based upon the responses from the developer's survey, and on literature references, a list of which is provided at the end of each write-up. The acronyms used in the catalog are those assigned by the developers. When a developer had no preferred acronym for his or her method, the authors chose an acronym to serve as an identifier of the method in this catalog.

The first section of the description provides background information which includes a synopsis of the method and a brief history including a list of other methods from which the current method evolved.

The next section presents a summary description of the method, with information on such topics as approach, the sequence of events followed in using the method, the components of the method, and where these components are used in the software development process.

The third section, "Technical Aspects", reports estimates of the number of organizations using the method and of the number of delivered systems. Opinions of the developer are reported regarding appropriate application areas, and the appropriate size of application for which to use the method. Further information on applicability is provided related to the method's ability to incorporate particular characteristics of the target system, and any relationship of the method to programming languages. Also, the modes of expression required by the method are reported, as well as any mapping rules prescribed for translating from one mode of expression to another, and the way that the method assists in translation across phases of the software process. Further, techniques for analysis and requirements clarification are discussed, as well as aspects of the method which address changes in the requirements, maintenance, portability, and reusability of the resulting software.

The fourth section, "Project Control and Communication", discusses the way the method addresses project management activities, as well as the communication channels provided by the method within the technical team, for management, and for understandability with the software client. The means by which the method involves the software client in the development process is reported, as well as the software documentation associated with the method. Quality assurance issues such as verification and validation techniques, as well configuration management, are presented.

The fifth section, "Ease of Use", provides information on technology insertion, including the developer's opinion of the essential concepts that must be understood and the minimum qualifications of a team leader for successful use of the method. Also given are estimates of learning times, and the training available for learning the method. The facilities for incorporating automation are summarized, including an overview of specific support tools.

The sixth section, "Acquisition Factors", lists cost estimates, presents the required hardware and software configuration needed to support the method, and gives contact information. If the acronym used in the catalog to represent a method is not a standard product name, this will be noted under the contact information section.

The final section, "References", lists books or articles which should be consulted by readers who wish to learn more about the method. These references are also listed under "Method-Specific Literature" in the bibliography which appears in Appendix A.

Comparing and Contrasting Methods

In Chapter 6, "Summaries of Responses to Selected Survey Questions", the reader is provided with a convenient format for contrasting methods. The authors have selected a set of questions whose format provided a ready basis for comparison of responses. For each of the selected questions, the responses of the developers have been summarized in tabular form. Topics which have been taken up in this chapter include development activities addressed and approaches used by the method, extent of usage of the method, recommended size of application with which to use the method, appropriate application areas for use of the method, and the ability of the method to address specific constraints of the target system. Additionally, this chapter addresses modes of communication used by the method, support provided by the method for changeability, for verification, for project management and for documentation, assistance available for training in the method, and estimates of times needed to learn the method. Further topics include the relationship of the method to software process paradigms and to programming practices or concepts, and the developer's opinion of minimum qualifications needed for successful use of the method.

The results reported in these tables represent the responses of the developers. The authors have not attempted to evaluate these responses in any manner, but do provide the reader with caveats for how he or she may best interpret these responses.

1.4 CONCLUSIONS AND OBSERVATIONS

The data for the second edition of the catalog was based solely on the responses provided by developers to survey vehicles, with reference to technical literature for supplemental information.

The definitions given in this catalog and the set of characteristics chosen to describe methods may not meet with universal agreement. The experience of creating the catalog convinced the authors that these choices were reasonable. Accordingly, the authors conclude that (1) the broad criteria chosen to select methods were appropriate, and (2) useful information about methods can be communicated by using a largely descriptive approach.

Several observations have been drawn by the authors as a result of the effort to produce this catalog. First, methods developed more recently appear to be more prescriptive, are being tailored to specific application areas, and/or attempt to involve the client more closely in the development process. Secondly, methods which use formal justifications to show correctness are prominent in the European community, and are finding increased importance in the United States. The authors have also observed that the European methods, both current and emerging, tend to have more demanding requirements for education than methods originating in the United States. A fourth observation is that there is a movement towards object-oriented approaches. Finally, structured analysis is a progenitor of many current methods: that is, many methods, both those that have been used extensively and those which have emerged recently, prescribe some form of structured analysis for concept exploration and system specification.

This page is intentionally blank.

CHAPTER 2

AN OVERVIEW OF BASIC CONCEPTS

2.1 INTRODUCTION

Terminology and Complexity

Software engineering encompasses many concepts, from project management through system design and software coding, to maintenance. Consequently, the terminology associated with the field is extensive and in a state of change, making descriptive clarity somewhat difficult. The concepts associated with the software process involve both the study of distinct ideas as well as their interrelationships. As in all fields, there is a need to categorize concepts in order to attain some level of abstraction.

Currently, terminology in software engineering is not universally agreed upon - terms are overloaded, unnecessarily synonymous with others, or represent concepts which have not yet stabilized. In addition, proper terms to convey distinctions between overlapping concepts may not exist. Moreover, there are attributes of software systems which are very difficult to understand and/or measure. Consider, for example, terms like maintainability, reusability, portability, robustness, testability, and changeability. Casual use of such a term could give the mistaken impression that the concept itself is well-defined.

In the sections which follow, an attempt has been made to provide the reader with a brief overview of software process concepts pertinent to this catalog. The reader is also provided with the authors' observations on the similarities and differences between some of these concepts. Because of the extensive amount of software process literature, the authors are not attempting to make an in-depth presentation. Instead, the authors' focus is intended to highlight the differences between concepts, guiding the reader to other sources. It is hoped that for those readers who are new to this field, this chapter provides some clarification of terminology. For all readers, it is the authors' intent to establish a basis of terminology and concepts used in the catalog. The reader is encouraged to consult the general bibliography in this catalog to gain a more thorough acquaintance with software process research.

Definitions

In this section, definitions of key terms used in the catalog are stated. They are provided so that the reader can get some sense of the authors' understanding about methods and related ideas. Furthermore, the definitions set a standard for the terms used throughout the catalog. Three important definitions stated in Chapter 1 are repeated here.

> Method: A definite, established, logical, or systematic plan. The steps and purposes have been thought out beforehand in detail. A scheme is a method when it guides the user to a predictable result given an appropriate set of starting conditions.
>
> Approach: A way of beginning or managing an effort; a way of analyzing, planning or directing a project; a way of conducting operations. A scheme is an approach when it suggests ways to identify goals initially and/or suggests, at an abstract level, ways to proceed toward goals.

Environment: An integrated collection of tools supporting an approach. The components of an environment are designed to reduce the effort required to carry out the software development process whether they are used individually or in combination.

The following definitions were offered by the steering committee of the March 1985 International Workshop on the Software Process and Software Environments [IWSP85].

Software Process: The collection of related activities, seen as a coherent process subject to reasoning, involved in the production of a software system.

Software Process Model: A software process model is a purely descriptive representation of the software process. A software process model should represent attributes of a range of particular software processes and be sufficiently specific to allow reasoning about them.

In the sections that follow and throughout this catalog, the authors strive to use terms in a manner consistent with the above definitions. The reader is encouraged to join with the authors in this effort to be consistent and careful in the use of terminology associated with the software process.

2.2 SOFTWARE PROCESS ABSTRACTIONS

2.2.1 Introduction

Models and approaches are abstractions which help to conceptualize the process of software development. A software process model offers a means of representing the components of the development process, while a software approach offers a means of specifying how activities are to be performed in accordance with some model.

The model concept, taken at a high level of abstraction, and applied to software development, is exemplified by the concept known as the "software life-cycle". This life-cycle notion captures the idea of software existing beyond the design and programming stage of development. The life-cycle concept implies that software exists even in the conceptual stages of problem definition, that it needs to be maintained after formal development has been completed, and that it functions until retirement. This idea has served to expand the boundaries of what software is by emphasizing the importance of factors outside of the development environment, such as the software client and software support centers for maintaining systems. In spite of the generality of this idea, however, it is the authors' opinion that the use of the term life-cycle has often been restricted to a single model. Accordingly, in this catalog the authors have elected to use the term "software process" in order to convey a more broadly based concept.

Models, be they of the software development process or of the software itself, are constructed with the model builder's bias of what is important. This bias determines what set of concerns will receive greater visibility. One of the more recent concerns in many of the process models is the concept of "assurance", that is, that the software product matches the client's intent. A model may deal with this concept by including the role of the software client in validating the system. Another model may incorporate this concept through a structured formalism which ties requirements to the end-product. Assurance may also correlate to other considerations such as the type of system to be developed, the technical orientation of people involved in the development effort, or the tools used in the effort. Consequently, the appropriate considerations need to be represented in the process model.

Approaches to software development also cover a range of abstraction. An example of a high level approach might be a corporation's strategy for managing a project, including the choice of environments and methods to be used for software development. At a lower level, the software decomposition strategy proposed by a method may also be called an approach. Such a decomposition strategy is, in turn, based upon a model of the target system as seen from a technical viewpoint. The model itself is biased by the parts of the system given a high priority by the model maker. Thus, some models assign data a primary position while other models choose to focus on the tasks the system must perform.

An opinion exists among some experts that there are benefits to having different views of the target system. These experts advocate a strategy of using different models simultaneously to produce multiple views. On the other hand, the same goal may be attained by linking one distinct model to another, creating a hybrid. For example, some models that are data oriented have been modified to incorporate control constructs.

At times, it is difficult to partition software process abstractions in a manner that distinguishes an approach from the underlying model. The attempt to conceptualize certain process models requires some elaboration of the approach to be followed. The model represents "what" is to happen in the software development process; an approach specifies "how" this is to be accomplished. It is not always easy to separate the two. Examples of this are the rapid prototyping model, and the spiral model. On the other hand, approaches are proposed for which no related process model has been explicitly elaborated.

In summary, abstraction is required in order to reason about the software process. On the other hand, the establishment of clearly defined concepts has proven difficult to achieve.

2.2.2 Software Process Models

An interesting insight into the concept of models and their relevance to software engineering is quoted by M. Lehman from a reprinted article which originally appeared in "The Encyclopedia of Ignorance":

> In general, as knowledge and understanding of an artificial, man changeable, system increases, we attempt individually and collectively to modify the behavior of that system.... The resultant configuration is and must be treated as a different system which requires a new model to represent it. Thus artificial systems and their models appear to be essentially transitory, 'continuously evolving'. [Lehm85]

In the following paragraphs, the models presented exhibit various ways of viewing the software development process; these viewpoints, in turn, contribute to the formulation of other models. For a relevant collection of articles on this topic, see "New Paradigms for Software Development" [Agre86].

The Waterfall Model

The Waterfall model is one that many view as the classic description of the software process. This model was first introduced by W. Royce [Royc70]. This model divides the process into the following phases: system requirements, software requirements, analysis, program design, coding, testing and operations. Each phase is conceived in terms of inputs, processes, and outputs. In this model, it was intended that the software process proceed through the above sequence of steps with iterative interaction between phases confined to successive steps. Experience showed, however, the need for more interaction between non-successive steps. Variations of the model which allow for such interaction have been proposed.

Though there has been much criticism of the model, R. Pressman states:

> ... the classic life cycle paradigm has a definite and important place in software engineering work. It provides a template into which methods for analysis, design, coding, testing, and maintenance can be placed.... While it does have weaknesses, it is significantly better than haphazard approach to software development. [Pres87]

The Spiral Model

The Spiral concept has evolved at TRW under the leadership of B. Boehm. The model involves multiple iterations through cycles with the intent of analyzing the results of prior phases and determining risk estimates for future phases. At each phase, alternatives are evaluated with respect to the objectives and constraints, forming the basis for the next cycle of the spiral. Each cycle is completed with a review involving relevant parties.

Boehm states:

> The model reflects the underlying concept that each cycle involves a progression that addresses the same sequence of steps, for each portion of the product and for each of its levels of elaboration, from an overall concept-of-operation document down to the coding of each individual program. [Boeh88]

The Prototyping Model

The Prototyping model advocates the early development of components representing the eventual system. Often these components represent the user interface to the system. A skeletal implementation of this interface is developed with the intent of providing an opportunity for feedback from the software client before the final system is specified and designed.

In an overview of prototyping, W. Agresti states:

> A software prototype is an executable object for which the users and developers have different expectations than they have for the corresponding delivered software product.
>
> "The expectations for prototypes often include less functionality or poorer performance than the delivered product will provide. [Agre86]

While the clarification of the user interface is one goal, prototyping may also be employed as a concept within the context of another model. In this case, the second model of the software process may regard prototyping as but one component of the process, to be used to clarify the behavior of the system at an early point in development.

Incremental Model

In the Incremental model, an initial subset of the system is fully developed. Then in successive steps, more elaborate versions are built upon the previous ones. The architectural design of the total system is envisioned in the first step, but the system is implemented by these successive elaborations. The software is developed in increments which represent degrees of functional capability.

The advantages of incremental development over either a pure top-down approach or prototyping are:

> The increments of functional capability are much more helpful and easy to test than the intermediate level products in level-by-level top-down development. The use of the successive

increments provides a way to incorporate user experience into a refined product in a much less expensive way than the total redevelopment involved in the build-it-twice approach. [Boeh81]

Operational Model

Behaviors particular to the problem domain are modeled and simulated in the beginning stages of the operational model, in order to explore with the software client the way and order in which events happen. This exploration is made possible with the construction of an operational specification of the system. The concern at the specification level with how the system behaves is in contrast to models whose specifications define the system in terms of a black box mapping inputs to outputs.

While the behavior of the problem domain is emulated in the specification, the software structures that will eventually be used in the actual system to produce this behavior are determined later in the development process.

P. Zave describes the operational approach as follows:

> During the specification phase, computer specialists formulate a system to solve the problem and specify this system in terms of implementation-independent structures that generate the behavior of the specified system.... This description may make an operational specification sound like a design, but it is not. First of all the structures provided by an operational specification language are independent of specific resource configurations or resource allocation strategies ... while designs actually refer to specific runtime environments. [Zave84]

Transformational Model

The Transformational model starts with a program specification and ends with a program, not unlike the Waterfall model. The difference is that in the former, progress between the two points is made through an automated series of transformations.

Definitions of terms associated with this model are given by H. Partsch and R. Steinbrueggen:

> 'Transformation rules' are partial mappings from one program scheme to another such that an element of the domain and its image under the mapping constitute a correct transformation.... 'Transformational programming' is a software process of program construction by successive applications of transformation rules. Usually this process starts with a (formal) 'specification', that is, a formal statement of a problem or its solution, and ends with an executable program. [Part83]

According to Agresti [Agre86], the benefits associated with this model include:

- Reduction of the labor intensity of software production through the automated transformation;
- Assistance in preserving correctness through the application of formal transformations;
- Replacement of final product testing by verification of the program specifications;
- The ability to produce the desired transformations through a combination of small units of specialized programming knowledge.

Fourth Generation Techniques

Recently, approaches to software development have emerged which make extensive use of fourth generation tools. These tools allow specification at a high level with code automatically generated based upon some formalized specification. The approach involves specification in a notation which captures functionality, or in a language which is close to a natural language. The goal is to produce the software from a high-level specification.

The approach utilizes an environment which supports some combination of capabilities such as the following: nonprocedural database query languages, spreadsheet functions, report generators, screen definition/interaction capabilities, automated code generation, and high-level graphics facilities.

The major steps in the fourth generation techniques involve definition of the requirements, choice of a design strategy, implementation using fourth generation languages, and production of the final system. Several iterations occur through these steps to allow the developer and client to clarify the precise requirements for the system.

In summarizing the current state of this approach, Pressman states:

> With very few exceptions the current application domain for 4GT is limited to business systems applications, specifically, information analysis and reporting keyed to large data bases....
>
> "Preliminary data collected from companies using 4GT seem to indicate that time required to produce software is greatly reduced for small and intermediate applications and that the amount of design and analysis for small applications is also reduced.
>
> "However, the use of 4GT for large software development efforts demands as much or more time for analysis, design, and testing (software engineering activities) thereby negating the substantial time saving achievable through the elimination of coding. [Pres87]

2.2.3 Software Approaches

In the following sections, the authors describe some basic schema by which software is conceived. An examination of different methods shows that these schema, or variations thereof, occur frequently.

Structured Approaches

Structured approaches have been proposed for both analysis and design. In the analysis phase, hierarchical and functional relationship between objects and activities are identified. At each level in the decomposition, components of the system are characterized in terms of the parent component, input, output, control, activity, and mechanism supporting the component.

Classic structured analysis was introduced by T. DeMarco, based upon the use of data flow diagrams [DeMa78]. Data flow diagrams model the process in terms of data flows and transformations; they form a network showing data entering as input, proceeding through a transformation process, perhaps in conjunction with other data, and becoming output. Additionally, data flow diagrams provide a distinct representation of the external entities involved in the system. For example, the people or job functions may be represented directly, such as "customer" or "billing department".

L. Peters summarizes structured analysis as follows:

> Requirements definition and logical design are linked or integrated into a single phase called structured analysis ... user participation is also employed to ensure that the results of analysis do reflect the customer's needs based on the present situation (current physical model), its abstract equivalent (current logical model), and the new system or solution model (new logical model). [Pete81]

Structured design proposes to map the flow of data from its problem domain into its software structure. The steps of structured design involve characterization of the data flow through graphical representation, identification of the various transform elements, assembling these elements in an hierarchical program structure, and refinement and optimization.

A key component of structured design involved the evaluation of the modular structure of the design relative to the concepts of coupling and cohesion. W. Stevens, G. Myers, and L. Constantine stated:

> Coupling is the measure of the strength of association established by a connection from one module to another. Strong coupling complicates a system since a module is harder to understand, change, or correct by itself if it is highly interrelated with other modules. Complexity can be reduced by designing modules with the weakest possible coupling between modules. [Stev74]

The related concept of cohesion is a measure of the single-purposeness of a module. Modules with a high degree of cohesiveness not only are understandable but are excellent candidates for reuse.

In assessing the wide popularity of structured design, L. Peters states:

> Structured design has gained wide popularity for two primary reasons. One is that it allows the software designer to express his perception of the design problem in terms he can identify with: data flows and transformations. The notation with which he expresses these flows is simple, easy to use, and understandable by management, customer, and implementor.
>
> "The other primary reason for this method's popularity is that it provides the designer with a means of evaluating his (and others') design, serving as a sort of benchmark against which to measure his success or progress. In this regard, the method is unique. In fact, if this method consisted of nothing more than the design evaluation concepts of coupling and cohesion, structured design would still be a significant contribution to the software field. [Pete81]

Object-Oriented Approach

The Object-Oriented approach to the software process is one in which models of entities are constructed as self-contained components. The system is defined by the interactions and behavior of these components. An important aspect of the design process patterns the behavior of the model so that it is "visible" only where there are interactions expected with other entities. Thus, the behavior that is self-contained is undetectable by these other entities.

In this approach, the concept of type is extended to class in which a model inherits or extends the characteristics of other models. Further, program entities may refer to objects of more than one class, a characteristic known as polymorphism.

A system's behavior is patterned upon the behavior of objects manipulated by the system, not the "function" of the system. The point is to address what it is that the system acts upon, rather than what the system does.

G. Booch says that:

> ... object-oriented development requires certain facilities of the implementation language. In particular, we must have some mechanism to build new classes of objects (and ideally, a typing mechanism that serves to enforce our abstractions). It is also the case that object-oriented development is only a partial-lifecycle method and so must be coupled with compatible requirements and specification methods. [Booc86]

Entity-Relationship Approach

This approach uses the Entity-Relationship (ER) model [Chen76] to categorize the information from the real-world problem domain. It recognizes that the database, as well as the code, needs to be considered at both a logical and physical level. Such information is conveyed by defining the entities in the domain, the interrelationships of those entities, and the attributes possessed by those entities. Ultimately, these concepts must be mapped into a schema which is implementable on a database management system.

In the early stages of development of systems which involve an underlying database structure, the ER model is often used as a means of conceptualizing information at a high level. In [Davi83], a perspective is given on the origins and use of the ER approach, as well as some reasons for its increased popularity since 1975.

Event-Oriented Approach

This approach is characterized by the concept of stimulus- response, where events are the stimuli to the system, and responses are comprised of actions taken by the system and the resultant outputs. Event-orientation builds the system based upon the types of events the system is likely to encounter.

Stepwise Refinement

In a seminal paper published in 1971, N. Wirth proposed the concept of stepwise refinement, a top-down design strategy. The process starts at a high level of abstraction, and incorporates details through a sequence of elaborations. This method of program decomposition parallels the process of partitioning and refinement that is frequently used in the analysis of requirements.

Wirth summarizes the refinement steps as follows:

> In each step, one or several instructions of the given program are decomposed into more detailed instructions. This successive decomposition or refinement of specifications terminates when all instructions are expressed in terms of an underlying computer or programming language, and must therefore be guided by the facilities available on that computer or language. The result of the execution of a program is expressed in terms of data, and it may be necessary to introduce further data for communication between the obtained subtasks or instructions. As tasks are refined, so the data may have to be refined, decomposed, or structured, and it is natural to 'refine program and data specifications in parallel'. [Wirt71]

During the refinement process, a notation which is natural to the problem should be used as long as possible. Ultimately, the implementation language will determine the direction in which the notation must evolve.

Wirth further states that each refinement step involves incorporating design decisions, such as efficiency, clarity, and regularity of structure. Various aspects of design alternatives must be weighed. At times is is necessary to revoke early decisions, and to return to an earlier step. If done carefully, stepwise refinement provides a modularity which facilitates later change.

2.2.4 Associated Principles and Practices

Frequently used practices supporting development that do not constitute methods or approaches are described below. They are not methods or approaches because they do not specify how to develop the system; rather, they merely provide guidance about how to structure the software.

Information Hiding

The concept of information hiding was introduced by D. Parnas. It proposed a way of decomposing a system to allow for changeability, comprehensibility, and the possibility for parallel development of system components. This is accomplished by hiding internal design considerations from other modules, and by avoidance of shared data and the modification of data "owned by" other modules. Furthermore, the knowledge of how data is implemented is hidden within the module to which the data belongs.

Parnas states that when information hiding is used, "modules no longer correspond to steps in the processing.... Every module in [such a] decomposition is characterized by its knowledge of a design decision which it hides from all others." [Parn72]

Structured Programming

In 1969, E. Dijkstra's article, "Structured Programming" [Dijk69] drew the attention of the software community to the concept of structured programming. The goal of structured programming is to achieve program verification in a formal manner.

Initial focus was on the limited set of control structures proposed for coding. These structures feature a single- entry/single-exit characteristic that facilitate an understanding of the control-flow of a software program. Though not appearing in this initial article, concepts associated with stepwise refinement and top-down design were also being incorporated by Dijkstra into the concept of structured programming. These concepts were elaborated in [Dijk72].

In looking back at the beginnings of structured programming, H. Mills states:

> ... Dijkstra's first article on structured programming did not mention syntax, typography, readability, stepwise refinement, or top-down development. Instead, his main argument for structured programming was to shorten the mathematical proofs of correctness of programs! That may seem a strange argument when almost no one then (and few now) bothered to prove their programs correct anyway. But it was an inspired piece of prophecy that is still unfolding. [Mill86]

2.3 RELATIONSHIPS BETWEEN METHODS, APPROACHES, TOOLS, AND ENVIRONMENTS

2.3.1. Introduction

The following sections discuss the problems in distinguishing the concepts of method, approach, tool, and environment. Observations are made on the similarities and differences between these concepts.

2.3.2 The Relationship of Method to Tools

Many of the methods surveyed in this catalog are embodied in an automated tool. In some cases, the method may be extracted from the tool without losing the essence of the method. In other cases, the tool and the method are so closely coupled that the method cannot stand alone.

The way in which a method develops influences its relationship to a tool. Methods which evolved without incorporating software tools may now receive support from a number of different tool vendors. Other developers treat automated support as a necessary component of their method, in which case the method and tool are likely to be synonymous.

Tools may also extend the scope of a method by addressing concepts beyond those originally considered by the method. In this case, it is difficult to distinguish whether the method has been redefined to encompass these additional concerns, or continues as but one part of a more elaborate scheme. In such a case, it can be difficult to determine whether the method incorporated the tool, or the tool subsumed the method.

This difficulty becomes even more complicated as a method which can exist independently disassociates itself from the tool in which it was first "incarnated". On one hand, the method developer may propose new aspects for the method that are not supported by the tool. On the other hand, the toolmaker may have his own ideas about the way development should be done, diverging from the developer's original ideas. A relationship still exists, but the ease with which the components of this relationship may be distinguished varies considerably.

All of the above reveals why it is difficult to clearly distinguish methods and tools. Though there are schema which are clearly tools, and other schema which are clearly methods, there is a broad spectrum over which these two types of schema merge.

2.3.3 The Relationship of Approaches to Environments

The correspondence between an approach and an environment parallels the correspondence between a method and a tool, but at a higher level of abstraction.

An approach may be likened to a framework in which specific methods can be incorporated. The approach is a general strategy for accomplishing goals; the particular methods incorporated in an approach specify the detailed steps for meeting these goals.

Tools are also specific resources for accomplishing particular tasks; tools are the entities which constitute an environment, which itself becomes a framework for the toolset.

An approach is necessary for addressing factors outside the scope of a particular method; the idea of coverage correlates more with approach than with method. Similarly, considerations associated with environments must consider broader issues that those addressed by an individual tool within the environment.

These considerations are necessary to provide a truly integrated environment which will support the total development process.

Both approaches and environments may be linked to particular methods and tools; they may also represent generic frameworks that can accommodate a variety of methods and tools. Some approaches make assumptions about the nature of the development environment, while existing environments imply a strategy which constitutes an approach. Because of this overlap in approaches and environments, it is often difficult to clearly distinguish one from the other. This difficulty with taxonomy parallels the problem of distinguishing methods and tools.

2.3.4. The Relationship of Methods to Approaches and Environments

Methods and tools may be seen as embedded components in a system characterized by an approach and an environment. Tools and environments comprise syntactic support for development. Methods and approaches support development from a semantic standpoint. Ideal support for the development process consists of a blend of these four components. This idea appears in the following statement from [Free82]:

> ... one cannot choose a tool, a management practice, or any other element of the total environment without considering that element in its relation to the other parts of the development system.

2.3.5 Summary

The sections above are intended to alert the reader to the multifaceted and complex issues which are associated with the concepts of method, approach, tool, and environment. In particular, it may be possible in the abstract to make precise distinctions among such concepts; however, in practice, such clear delineations cannot always be made. Because precise distinctions are difficult to make, for this catalog, the authors have assigned a wide interpretation to the term "method".

2.4 COMPARISON CRITERIA FOR METHODS

2.4.1 Establishing a Basis for Comparison

The information contained in this catalog is intended to provide a basis upon which the software engineering community can compare methods. The task of comparing methods can be addressed by examining the software produced using the method, by examining the impact of the method on the development process, or by examining features of the method itself. Unfortunately, a comparison of methods based upon an examination of either the resultant software, or the process of development, represents a formidable challenge.

Ideally, it would be desirable to show that use of a particular method results in more reliable software, or in more well-structured and maintainable software. Similarly, being able to demonstrate that use of a particular method results in greater productivity during the software process, or in better control of the process would be valuable. Attempting to confirm such cause and effect relationships requires that one demonstrate that the desired effect results primarily from use of the method, and not from the other factors associated with software development. Such factors include the composition of the development team, the environment provided for development, the organizational structure used to manage the project, the nature of the problem domain, and even the time-frame associated with development. Isolating the respective impact of each of these factors is an unresolved problem.

There is evidence to suggest that it is both possible and practical to create a basis by which methods can be compared by providing descriptive data about methods themselves. Furthermore, the comparison can be facilitated through use of an appropriate representation of the data. Such a basis is provided in this catalog through the use a uniform format in the description of individual methods, and by providing tables where specific features of methods are represented.

The underlying premise for this catalog is that an absolute comparison, or ranking, of methods is neither possible nor desirable. In a report on the assessment of methods, W. Wood and associates at the Software Engineering Institute made a similar assertion, stating:

> There is no such thing as an overall 'best' method for developing all software, only the method that will work best to help develop a system with particular characteristics and will blend with an organization's software development practices. [Wood88]

Thus, what is required is information by which a judgment can be made as to the suitability of the method to the problem domain, to the needs of the development team, to the available development environment, and to the organization's management practices.

As part of the task of creating this catalog, the authors developed a list of high-level comparison criteria associated with methods. The set of descriptive characteristics detailed in Section 1.3 are indicators of these high-level criteria. In the section following, each of these criteria is discussed briefly, and, for those methods listed in Chapter 3, the reader is directed to those places in the catalog where method-specific information associated with the criteria can be found.

2.4.2 Discussion of the Criteria

Coverage and Prescriptiveness

Two key criteria by which methods can be compared are coverage and prescriptiveness. By coverage is meant the set of activities of the software development process addressed by the method. By prescriptiveness is meant the level of detail which is supplied by the method insofar as providing direction on how to accomplish the various activities. Also associated with these criteria is the type of process model for which the method is suited, since the model influences which activities are addressed and to what degree of detail.

Information associated with these criteria is primarily found in the Description sections for the methods found in Chapter 3. Additional information is provided in Sections 6.1 and 6.3. Also, the legend in several of the tables of Chapter 6 is designed to specify the level of guidance provided by the method for the particular activity.

Robustness

By the criterion of robustness is meant the variety of problem domains for which the method is applicable. In other words, is the method geared toward a specific type of problem, or can the method be used for different types of applications? Associated with this criterion is the type of application and the size of application for which the method is intended. Additional related descriptive characteristics include the extent of use of the method, and how suitable the method is to the programming language in which the software system is to be developed.

The individual method summaries in Chapter 3 contain information associated with robustness under the sections entitled Applicability and Usage. Additionally, the Target Constraints sections contain information regarding the ability of a method to address specific features of the target system, such as timing or spatial constraints, fault tolerance, and security of access. The presence or absence of capabilities such as these may well dictate which problem domains the method can address. The reader is also referred to Tables 4, 5, 6 and 8 in Chapter 6.

Expressiveness

The criterion of expressiveness refers to the facilities provided by the method to represent the evolving system. Related to this criterion are the modes of representation used by the method along with the support provided by the method for communication among members of the development team, with the client, and with management.

Information associated with expressiveness has been provided in the sections of Chapter 3 entitled Modes of Expression, and Communication Channels. Summaries of survey responses for the modes of representation used by methods are provided in Sections 6.9 and 6.10.

Analyzability and Stability

Important criteria by which to compare methods include those features which assist the developers in designing complex systems, and those features which assist the development team in contending with the inevitable changes which will occur during the development of a large system. In particular, it is expected that there will be changes in requirements and changes in design decisions. Analyzability is that criterion which addresses the support provided for the activities of analysis and design. Stability involves the capability to continue to employ the method in spite of changed requirements or design modifications.

Achieving a clear comparison of methods based on these criteria requires the actual experience of using a method. Nevertheless, some level of comparison can be achieved by examining what aspects have been provided in the method to address these criteria, and such information has been provided in this catalog. These aspects are described in the section of Chapter 3 entitled Techniques for Analysis and Requirements Clarification. Additional information is provided in Tables 2, 11, 12, and 14.

Correctness and Effectiveness

The criterion of correctness involves the reliability of the software product produced using the method and the conformity of the product to the client's requirements. The effectiveness criterion involves the total quality of the resultant product. In addition to reliability, effectiveness is associated with how well the product software system is structured and the quality of the documentation for the system. In essence, an effective method will produce reliable and maintainable code.

Achieving a comparison of methods based on these criteria requires that one examine the product system. The difficulties of establishing a clear cause and effect relationship for such criteria was discussed above. It is possible, however, to list for methods those descriptive characteristics which assist in developing a quality system. It is this latter type of information which is provided in this catalog.

With regard to the criteria of correctness and effectiveness, the reader should consult in Chapter 3 the sections on Quality Assurance, and on Other Technical Aspects. With respect to effectiveness, additional appropriate information can be found in the sections associated with documentation and with recording decisions. In Chapter 6, Tables 7, 12, 15, and 16 contain related information.

Manageability

The criterion of manageability refers to the support provided by the method relative to planning, controlling, and monitoring the software development process. Assistance for the various aspects of project management provides a basis upon which to judge this criterion. Of related concern are those features of the method which assist in the preparation of required documentation for the system.

Information associated with this criterion is presented in the sections of Chapter 3 entitled Project Management, and Documentation Formats. Summaries of responses to questions related to this criterion are presented in Sections 6.13 and 6.16.

Productivity

The criterion of productivity refers to those aspects of a method which facilitate the process of developing software. Establishing that use of a particular method results in software being produced in less time or with less effort would, without doubt, be of great value. The difficulty of establishing control on the other factors of the process so as to claim a cause and effect relationship was discussed above, and may well be an impossible task. Accordingly, for this catalog, information is provided on those features of a method which are intended to facilitate the development process.

Foremost among features associated with productivity are the automated facilities available for use with the method. It is recognized that most of the other features of methods may also assist in reducing the effort needed to develop software. For example, this effort can be lessened by the availability of modes of representation which ease the analysis task, or by features which assist in the early detection of inconsistencies. For information appropriate to this criterion, the reader is referred to the Automated Facilities Sections of Chapter 3. The tables found in Sections 6.9, 6.10, 6.13, 6.14, 6.15, and 6.16 also contain information about automated support.

Ease of Adoption

The criterion of ease of adoption addresses those aspects associated with the introduction of a method into a development organization. Descriptive characteristics related to this criterion include the available training, the hardware/software configuration needed to support the method, and the cost of acquisition. Additionally, it is important to know what type of education and experience is required of the development team in order to gain proficiency in the use of the method.

Information associated with this criterion is provided in Chapter 3 in the Technology Insertion sections and the Acquisition Factors sections. In addition, Tables 17, 18, and 19 contain information related to this criterion.

2.4.3 The Catalog as a Resource

In concluding this section, the authors reiterate how the information provided in this catalog should be viewed. The information focuses on descriptive characteristics of features directly associated with methods. The information is based on responses from method developers obtained through the use of a questionnaire, and on method-related literature. The information is reported in a format intended to provide the reader with a basis upon which to compare methods.

Thus, by using the list of comparison criteria as a guideline, and the descriptive method-specific information about characteristics as a resource base, the reader is in a position to judge how well a particular method meets his or her current needs.

This page is intentionally blank.

CHAPTER 3

METHODS CURRENTLY IN USE

This chapter contains descriptions of methods currently in use. Among these are methods which are available commercially, methods which are company proprietary and available to the Government, and methods which are company proprietary and unavailable to the Government. In some instances, methods which are part of a company's resources are not being marketed but are available to the development community. Finally, certain methods may be in the public domain.

For the methods in this chapter, developer's questionnaires were distributed and responses received. Information presented is derived largely from these questionnaires and from technical references supplied by the developer. A sincere attempt has been made to include in this chapter all methods for which a developer survey was returned. In addition, extensive efforts were made to obtain survey responses from developers of other known methods.

Each of the methods described in this chapter is presented again in the tables in Chapter 6. In some sections of the method writeups, references have been made to the corresponding tables. This has been done primarily to avoid presentation of information in the writeup that was intended for presentation in tabular format. The reader is encouraged to use the tables to gain additional information about a particular method.

In reading the following descriptions, the reader should keep in mind that each description is a temporal "snapshot" of the method as of the publication date of this catalog. Many of these methods are undergoing change and their representations can be expected to differ in the future from what is presented here.

3.1 **ABDLSLCM** -- Ada Based Design Language System Life Cycle Methodology

3.1.1 **BACKGROUND**

Synopsis

ABDLSLCM is based upon certain characteristics (e.g., modularity, data expression, scope) of the Ada language. These characteristics are combined with the use of a Program Design Language (PDL) and extended to the earlier life cycle activities of requirements and design specification. The Ada Based Design Language (DL) statements are intended to formalize the ambiguous English language of the requirements or design documents.

History

ABDLSLCM was developed by Robert Weissensee and was first used for the development of a deliverable system in 1986.

The Program Design Language (PDL) component of ABDLSLCM originated with S. Caine and E. Gordon. Methods whose graphical diagramming techniques are compatible with the method include Structured Analysis/Structured Design and the Hatley/Pirbhai method.

3.1.2 **DESCRIPTION**

ABDLSLCM, Ada Based Design Language System Life Cycle Methodology, is based on a design language approach which specifically addresses the activities of requirements definition, system specification, and system design. The statements of this language are designed to be used for tracing requirements, generating documentation, and facilitating communication among people involved in the project. The language statements are structured in a format that can be utilized by tools for generating documentation to assist in the above functions. Associated with the method is a structured data base which can distribute current technical data to dispersed development teams.

Specific directions are available for tailoring the method to DOD-STD-2167A. Documentation levels of the method correspond to required Data Item Descriptions (DIDs) in the standard. A number of activities are iteratively performed at each documentation level over the course of the development process. These activities include:

- creation or updating of a program or project data base;
- annotation and definition of data;
- assignment of traceability information;
- decomposition of requirements or functions to separate paragraphs or subfunctions;
- development of context diagrams or functional hierarchy charts;
- analysis of risks;
- analysis of quantitative measures;
- performance of walkthroughs and formal reviews;
- baselining and generation of the document for that level.

3.1.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

The developer stated that the method is well-suited for use on applications involving embedded systems or process control, time critical or real-time processing, systems programming, data processing, and large scale simulation or modeling. Examples of systems developed using the method are a commercial real-time petrochemical tank leakage detection system, a computer-aided design language tool set, and a program management data base reporting system.

Between 5 to 20 delivered systems were estimated as having been developed using the method, in as many organizations. The method is intended for use on projects of all sizes; it has been used on small and medium-sized projects. The implementation languages most frequently used in conjunction with this method are Ada and Machine_Code_80386.

Target Constraints

ABDLSLCM addresses a number of target system requirements. When there is a requirement for timing constraints, quantitative timing estimates are made and later refined at various design stages, and compared to actual compiler and/or assembler code timing analyzer numbers at the coding and Computer Software Unit (CSU) stages. Spatial requirements are addressed in the method by estimating processor, memory and secondary storage, adding design language statements for data elements, verifying actual data usage refining the estimates, and comparing actual compiler or assembler code usage numbers.

Several of the method's design statements correspond to Ada language features found in Chapter 13 of ANSI/MIL-STD-1815A. These are MACHINE_CODE and utilization of Ada's representation clauses for addressing special features of the target hardware architecture, and OTHER_LANGUAGE, corresponding to Pragma INTERFACE, for addressing special features of the target operating system.

The method addresses concurrency issues by making estimates of concurrency requirements during system design. Tasks, task type structuring constructs, and assembler concurrency processes are used throughout the design and summarized in separate document sections at each level of decomposition. Fault tolerance issues are handled by specifying the exception processing that should occur at a particular level. Missing exception processing can be flagged by automated analyzer tools.

Modes of Expression (Tables 9,10)

The method requires several modes of textual representation: specified documentation templates, narrative overviews of modules, program design language, and formal specification language. Required iconographical modes are data-flow diagrams and hierarchy charts. The method encourages transaction timing diagrams. Automated support is provided by the method for all of the above modes.

The primary mode of expression in ABDLSLCM is an Ada based design language which maps directly into a number of iconographical diagrams in the following ways:

- design language program unit identifiers are the same identifiers used in the hierarchy diagrams;
- design language unit identifiers and their formal parameters including mode are the same identifiers and directional flow indicators that are used in the Data Flow Diagrams (DFDs);
- the hierarchy diagram identifiers are the same identifiers used in the DFD bubbles;

- the flowcharts can be generated from the program unit body section design language processing statements;
- the Transaction Timing Diagrams and Performance Models utilize the same data as the DFDs and selected timing records to help generate the diagrams or models.

By using an Ada based design language throughout the development process, the developer states that problems associated with phase or activity level transformations are eliminated.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

ABDLSLCM requires incremental or evoluationary development to clarify system requirements. Design reviews, code walk-throughs and design language analysis are also required techniques.

Other Technical Aspects

The method provides a means for tracking requirements throughout the development project. Each requirement is uniquely identified, made into a design language statement, categorized, and recorded in the functional/capabilities design language section that satisfies the requirement. Using the right tools, the developer stated that a software developer could thus determine the scope and magnitude of the requirement change as well as exert less effort to incorporate changes in requirements.

3.1.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

ABDLSLCM provides guidelines for many project management activities, and provides automated support for some of these, including support for quantitative metrics.

Communication Channels

Within the development team the method uses a combination of Ada based design language for textual representation and graphical diagrams to facilitate communication within the team. The diagrams are consistent with the design language and can be automatically generated from the text with the proper tools, which tools can also generate associated interface documents, e.g., Interface Requirement Specification, Interface Design Document.

Between the technical development team and management, the method uses the same representations as above. Associated with each of the method's structuring mechanisms, or Logical/Program Units, is a Preface section which can contain a number of information items which may be of interest to management, e.g., traceability, accountability, security, origin, resources. These information items may be requested by management throughout the various development activities. In conjunction with Airspace Technology's Program Management Forecasting, Tracking, and Reporting System tool, a variety of reports, graphs, and slides can be produced for use by management.

Communication between the client and the development organization is based on the textual and graphical modes listed above, at a level which can be comprehended by both Ada trained and non-Ada trained personnel. Moreover, requirements are recorded in the functional/capabilities sections that satisfy the requirements, which provides visibility to the client. A third aspect of communication with the client is the feasibility of doing rapid prototyping of selected system components, due to the production of compilable or assembly code as a by-product of the method's requirements and design phases. Finally, the method encourages close involvement of the client with the development organization throughout the life cycle, in terms of walk-throughs and informal as well as formal reviews.

Quality Assurance (Tables 12,14,15)

Prescriptive checking of interfaces are specifically addressed and provided with automated support by the method. The method provides guidelines for testing activities and automated support for test planning, test generation based on system requirements, and unit/integration testing.

Text files are analyzed for consistency and omissions within a given activity level. Consistency across a level is ensured by generation of associated documents, e.g., interface documents. Between activity levels, requirements are linked to logical/program unit section structures.

The method provides tailorable document formats for recording design decisions, problem logs, and change logs. It alsɔ provides a framework for configuration management activities, with automated support.

Documentation Formats (Table 16)

All documents required by the method are tailorable within the method, and most such documents are automatically generated based on data produced from other steps in the method.

3.1.5 EASE OF USE

Technology Insertion

The developer estimated that minimum qualifications for a development team leader's successful use of the method would be a bachelor's degree, three to five years of development experience, working knowledge of two programming languages, and experience working on two different software systems. Successful use of the method by an experienced developer requires that the developer understand the concepts of functional decomposition, modularity, and partitioning as well as concurrency and exception processing as they occur at all decomposition levels. In addition, the developer should be able to use an Ada based design language as a formal requirements specification language, and know how to distinguish and treat separately on-line transaction processing versus real-time processing.

The developer offers overview presentations, classroom tutorials, on-site consulting, a "hot line" service, user manuals, and periodic technical updates as means for training in the method. One day would be required for a project manager to acquire a basic understanding, ten days required for an experienced developer to learn to use the method's essentials, and two months required for an experienced developer to achieve the level of expert user.

Automated Facilities

ABDLSLCM provides automated support for a number of required and encouraged modes of textual and graphical representation. There is automated support for all the documents required to be produced by the method. Several verification and project management activities are assisted with automated support.

When the method is used in conjunction with RAMTEC's "Ada Based Design Language System Life Cycle Documentation Guide" and DL Tool Kit, a design language text analyzer is used to automatically ensure consistency and will report errors and omissions. The tool kit's Technical Assessment option provides for quantitative analysis of the documents being developed or maintained. Such quantitative analysis can be performed form the developmental or maintenance perspective and involves requirements, transactions, Logical/Program Units, and test procedures.

Airspace Technology Corporation provides a tool called the Program Management Forecasting, Tracking, and Reporting System. This tool supports managerial activities, e.g., forecasting, tracking, ad hoc reporting, slide preparation. An optional feature is a direct interface with the DL Tool Kit which provides automatic data updating and verification.

3.1.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

The Ada Based DL Tool Kit, which supports the ABDLSLCM, requires an IBM PC configuration.

Acquisition Costs

Contact the developer for quote on costs. The licensing policy for the DL Tool Kit encompasses various options with discount schedules.

Contact Information

RAMTEC Inc. 201-477-8248
727 Eastern Lane
Bricktown, NJ 08723 [Provider of method and tool]

Airspace Technology Corporation [Tool vendor]

3.1.7 REFERENCES

Contact the developer for publications by RAMTEC on the method.

3.2 ADM -- Ada Development Method

3.2.1 BACKGROUND

Synopsis

The developer characterizes this method as an iterative "design a little, code a little, test a little" method, founded upon both data flow-oriented and object-oriented approaches. The method specifically addresses almost all activities associated with software development and is tied to the use of Ada for expressing and communicating the design. A major aspect of the method is the production of compilable designs and tested code early in the life of a project. ADM is available for general use.

History

The principal architect of ADM is Donald G. Firesmith. An early version of this method was first used in 1985, on the U. S. Army's Advanced Field Artillery Tactical Data Systems (AFATDS). ADM is related to several pre- existing methods, including Booch's OOD, PAMELA II, and Extended Buhr Design Method. It also incorporates ideas from NASA's Generalized Object-Oriented Development, Bulman's Model-Based OOD, and EVB's OOD.

3.2.2 DESCRIPTION

ADM, Ada Development Method, addresses almost all activities associated with software system development. The method considers object-oriented development to be the paradigm promoting its most effective use, and is well- suited for use within the context of the incremental and recursive life-cycle models. ADM's essential concepts and programming practices include stepwise refinement, information hiding, process abstraction, abstract data-types, structured programming, genericity, and module coupling/cohesion.

The developer stated that the major aspects which differentiate the method from others directed towards the same application domain are object-orientation instead of functional decomposition and the fact that the method follows a recursive model instead of the Waterfall life-cycle model. In addition, ADM covers Requirements Analysis and Tasking Architecture, which, according to the developer, most other OOD methods do not address.

The steps of ADM consist of identifying "Assemblies" and "Builds", as well as scheduling and staffing the assembly development. In addition, the steps for each build consist of iteratively developing the relevant subassemblies from assemblies, testing the build, and releasing the build software and documentation to configuration management. Development of each subassembly involves Subassembly Requirements Analysis and Design, Subassembly Code, Test, and Integration, and Assembly Integration.

During Subassembly Requirements Analysis and Design, a number of activities occur. One of these activities involves initiating subassembly development by scheduling milestones and staffing the development teams. Another activity, analysis of subassembly requirements, includes storing the subassembly requirements in the subassembly Software Development File, stating the subassembly objective, reviewing of the subassembly requirements, and identification of all subassembly abstract objects. Also included are the development of an initial Object Diagram, analysis and organization of the subassembly requirements by abstract object, and development of a subassembly object-oriented Data/Control Flow Diagram. Other activities involve:

- creating a logical design for the subassembly,

- analyzing all abstract objects in the subassembly,
- identifying abstract object packages and generic packages,
- designing the tasking architecture,
- coding and compiling all package and generic package specifications,
- making additional design decisions (e.g., reuse, recursion),
- performing Subassembly Requirements and Design Inspection, and, as necessary,
- repeating operations to form new subassemblies.

The Subassembly Code and Test activities involve coding and compiling of all package bodies. Subassembly testing is planned, and subassembly test software is designed and coded. After initial test, the subassembly software is integrated. After the performance of Subassembly Code and Test Inspection, the subassembly is integrated into the assembly, and whatever deliverable documentation exists is updated.

3.2.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

The developer stated that the method is well-suited for use in applications involving embedded systems or process control, time critical or real-time processing, scientific or engineering processing, data processing or database applications, and large scale simulation or modeling. The method has been used on the U.S. Army's Advanced Field Artillery Tactical Data Systems (AFATDS). Less than five organizations have used this method, and there are less than five delivered systems that have been developed using ADM. Although the method is intended for projects of all sizes, it has been used on a large project only. Ada has been the implementation language most frequently used with the method.

Target Constraints

ADM addresses timing constraints imposed by the target system by means of timing diagrams. Concurrency issues are addressed with Buhr diagrams, timing diagrams, Petri nets, and task sequencing language. The method also addresses fault-tolerance issues with an exception handling architecture. The developer stated that, by being an object-oriented method, the design naturally maps well to hardware peripherals, so that when hardware changes, the impacted code is highly localized.

Modes of Expression (Tables 9,10)

The method requires a program design language and several iconographical modes of representation, including Petri nets, data-flow diagrams, control- flow diagrams, hierarchy charts, Buhr diagrams, and Firesmith diagrams. Mapping rules for translating from data/control-flow diagrams to subassembly OOD diagrams, and from object diagrams to data/control-flow diagrams, are prescribed by the method. Translation across phases of the software process is aided by use of Ada-oriented graphics and a compilable PDL that evolves into the deliverable code. The developer states that the object-oriented DFDs, or requirements, lead naturally into subassembly OOD diagrams, or design.

Techniques for Analysis and Requirements Clarification (Tables 11.12)

A recursive version of rapid prototyping, incremental or evolutionary development, and executable specifications are techniques required by ADM for clarifying system requirements. Required analysis and review techniques include data and control-flow analyses, design reviews, code walk-throughs, and Change Control Board review.

Other Technical Aspects

The developer reported that software produced with ADM via its object- oriented approach is more extensible because changes to requirements are better localized in the design due to lower data coupling. The early detection of inconsistencies and/or errors results from the iterative nature of the method, which produces compilable designs and tested code early. Assistance in identifying possible reusable components is given in the "Additional Design Decisions" step of the method. The developer also says that all code is naturally more reusable since the default packages implement Abstract Data Types (ADTs) and Abstract State Machines (ASMs).

3.2.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

ADM prescribes specific procedures for assessing complexity and tracking project progress. It also specifically addresses project planning, scheduling and/or manpower loading, and allocation of personnel to tasks.

Communication Channels

The aspects of ADM which the developer reports as being designed to facilitate communication between all parties involved in the software process are: graphics, verb/direct object PDL, and Ada. Involvement of the client is accomplished with a recursive type of rapid prototyping, in which the design and code are developed at an early stage. Since the design is compilable, the client gets a tested design as opposed to only one on paper.

Quality Assurance (Tables 12,14,15)

Test planning at one or more precise points in the software process and unit/integration testing are specifically addressed by the method. Consistency between the design and code is maintained through the use of Ada- oriented graphics and a compilable PDL that eventually becomes the code that is a result of the object-orientation of the DFDs which in the developer's opinion transform naturally into subassembly OOD diagrams.

ADM provides guidelines for configuration management. While requiring that a record be kept of specification/design options which were considered, personnel involved in decisions, and changes related to specification/design decisions, the method does not provide specific directions for recording these types of information.

Documentation Formats (Table 16)

Documents required by the method do not have their formats prescribed by the method.

3.2.5 EASE OF USE

Technology Insertion

According to ADM's developer, the minimum qualifications needed by a development team leader for successful use of the method were a bachelor's degree, one to two years of development experience, working knowledge of one programming language, and experience with one software system. In addition, the major theoretical constructs required for successful use of the method include abstract state machines, abstract data types, object abstraction, process recursion, and Ada.

Training assistance is available in the form of overview presentations, classroom tutorials, and on-site consulting by the vendor. Two days would allow a project manager to acquire an understanding of ADM's major features and benefits. An experienced developer would need five days to learn to use the essentials of the method, and three months to become an expert user.

Automated Facilities

The developer reported that any tools supporting DFDs, Buhr diagrams, Petri nets, and other iconographical modes required by the method would be useful.

3.2.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

High-resolution graphics workstations are appropriate for hosting the tools which provide support for the representation modes used by ADM.

Acquisition Costs

The cost for acquiring the method or for technical training is $10,000. A management overview costs $4,000. There is no licensing policy.

Contact Information

Mr. Donald G. Firesmith, President 219-456-9260
Advanced Software Technology Specialists
3418 Broadway
Fort Wayne, Indiana 46807 [Developer and provider]

3.2.7 REFERENCES

The main source of information on the method are the training materials in courses taught by Mr. Firesmith through Technology Training Corporation. A "white paper" on the method is being submitted by Mr. Firesmith for publication in a professional journal.

3.3 **AISLE** -- Ada Integrated Software Lifecycle Environment

3.3.1 **BACKGROUND**

Synopsis

This method is associated with ADADL, a Program Design Language (PDL) based on Ada, and a set of software tools designed to automatically produce Mil-Std documentation and unit test plans or procedures. The ADADL processor has analysis capabilities designed to detect errors in the logical description of the design. The method specifically addresses system design and system implementation activities.

History

ADADL PDL, which is an extension of Ada, was created by Dr. Thomas S. Radi. The AISLE family of tools was first used with respect to a deliverable system in 1986.

3.3.2 **DESCRIPTION**

AISLE, Ada Integrated Software Lifecycle Environment, is a collection of tools for addressing several activities of the software development process. It is used to describe, document, and assist the testing of the executable code. This method can be used to develop designs to be coded in languages other than Ada as well. The intent of the method is for the user to compose or change the design with PDL, then use the ADADL processor to analyze the logical description and consequences of the new or changed design, and finally derive the executable Ada code corresponding to the PDL. The developer states that the method provides specific procedures for conducting system design and implementation activities, and provides a framework or guidelines for accomplishing requirements definition, system specificaiton, software quality assurance, and project management activities. The method is compatible with almost any approach.

In addition to ADADL, AISLE includes a number of other tools. These tools and some of their capabilities are listed below:

- DocGen is an automatic Mil/DoD-Std document generator for producing documentation from the Ada source code;
- TestGen is a verification tool helping to prepare unit test procedures in conjunction with the PDL or source code;
- GrafGen is a graphic output display for the Ada program and interfaces, allowing user to work in a graphical environment with Buhr-like and Booch-like descriptions;
- ASE is an Ada/ADADL syntax directed editor;
- ARIS, Ada/ADADL Requirements Interface System, can extract information from a Real Time Structured Analysis (RTSA) database;
- AIEM (InfoGen), or ADADL Integrated Environment Manager, allows for on-line access to all design information;
- QualGen, Ada Quality Analysis and Metrics Generator, calculates over 60 quality metrics;
- RETT, Requirements Evaluation and Traceability Tools, provides forward and backward traceability of requirements and allows for seeing the effects of changing a requirement.

3.3.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

The developer rated the method well-suited for embedded systems, scientific or engineering systems, systems programming, image processing or pattern recognition, and large scale simulation or modeling. Delivered systems built with the method include weapons systems, navigation and guidance, and command and control systems. Between 21 to 100 delivered systems have been developed using the method in between 51 to 100 organizations. The method is intended for use on projects of all sizes and has been used as such. The implementation language most frequently used when coding systems developed with AISLE is Ada.

Target Constraints

Timing and spatial constraints of the target system are handled by production of reports of information entered by user for each program unit. Concurrency issues are handled through use of the Ada tasking models.

Modes of Expression (Tables 9,10)

Textual modes required and supported automatically are specified documentation templates, narrative overviews of modules, structured English, and program design language. Required iconographical modes are data-flow diagrams and control-flow diagrams. The method strongly encourages and provides automated support for Buhr diagrams and Booch diagrams. In addition to representation modes used, the method is considered inconsistent with a number of other modes; see Tables 9 and 10.

Mappings are provided for translating from requirements to top level design, and from design to code. A tool, ARIS, produces top level Ada design automatically from a database, such as one found with Teamwork or Excellerator.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

Incremental or evolutionary development is required for clarifying system behavior. Techniques for analysis and review include data-structure analysis, design reviews, and code walk-throughs.

Other Technical Aspects

By providing a way to cross-reference requirements to the program units that satisfy each requirement, the method assists in incorporation of changes in the requirements. Consistency is maintained among specification, design, and code by having the design and code in the same file, by generating quality analysis reports that check the design vs. code, and by a cross-referencing capability for the requirements.

3.3.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

Specific procedures and automated support is provided for analyzing risk, assessing complexity, and tracking project progress. Configuration management is required but not provided by the method.

Communication Channels

Communication is facilitated and coordinated between all parties involved in the software development process by the integrated tool support and the varied views of the development process that are produced. The client is involved by the method's encouragement of design reviews and the support for such reviews provided by TestGen's Design Review Expert Assistant.

Quality Assurance (Tables 12,14,15)

Specific procedures are provided for test planning. Also specifically addressed as well as provided with automated support are unit testing, field testing, and generation of test data. A quality assurance or test plan is automatically generated from previous steps in the method. In addition, the method provides automated recording procedures for maintaining a record of technical decision-making during the software development process; see Table 14.

Documentation Formats (Table 16)

A number of documents are required to be produced, the majority of which are automatically generated based on data produced from other steps in the method. See Table 16 for particular formats.

3.3.5 EASE OF USE

Technology Insertion

The developer estimated that a development team leader, for successful use of the method, would need as a minimum a bachelor's degree, no development experience other than some experience with one software system, and working knowledge of Ada. He also stated that good software design practices should be understood by an experienced developer in order to successfully use the method.

Training assistance includes hands-on demonstrations, overview presentations, classroom tutorials, on-site consulting by the vendor, on-line help facility, a "hot line" service, user manuals, a users' support group, and related publications from third-parties.

Estimates of learning times were given as two days for a project manager to understand the major features and benefits, four days for an experienced developer to learn to use the essentials, and two months for an experienced developer to achieve expert user level.

AISLE

Automated Facilities

The developer provides a number of automated tools which are integrated into AISLE. See the description section and Tables 9, 10, 13, 14, 15 and 16.

3.3.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

Appropriate configurations for hosting the tools of the method are:

- VAX and Micro Vax/(VMS, Unix or Ultrix)
- DataGeneral/MV-AOS
- Gould (SEL)/(MPX or UTX)
- Concurrent
- Harris
- HP800
- Sequent
- Pyramid
- Apollo
- HP9000/3xx.

Acquisition Costs

Tools in the AISLE *family* are priced individually according to host configuration. Training and consulting is provided by the developer at $2000 per day plus travel expenses. Licensing is per CPU or per node.

Contact Information

Dr. Thomas S. Radi 714-625-6147
Software Systems Design, Inc.
3627 Padua Avenue
Claremont, California 91711 [Developer]

3.3.7 REFERENCES

Further information is available from Software Systems Design, Inc.

3.4 BOX STRUCTURES -- The Box Structure Methodology for Information Systems Development

3.4.1 BACKGROUND

Synopsis

The box structure methodology is described as a complete, mathematics-based theory that extends software engineering principles to systems. Each part of the system under development is viewed at three different levels of detail: as a black box, a state box, and a clear box. Defining the parts of the system each in terms of these three data abstractions reduces the size of the analysis and design steps used in systems development [Hevn88].

History

First used in 1987 for the development of a deliverable system, this method is an extension of data abstractions, objects, usage hierarchies of data abstractions, and mathematical verification of programs and systems. The evolution of the method occurred at IBM Federal Systems Division. The principal architects are H. D. Mills, R. C. Linger, and A. R. Hevner.

3.4.2 DESCRIPTION

Box Structures, or the Box Structure Methodology for Information Systems Development, specifically addresses activities concerning requirements definition, system specification, system design, implementation, and software quality assurance. The method uses stepwise refinement and verification to produce a system design from the specification. The design is composed of small steps that allow immediate verification using mathematical derivation techniques that map one step into another.

The design process of the method is founded on three principles [Mill88]:

1. All data to be defined and stored in the system is hidden in data abstractions.
2. All processing is defined by sequential and concurrent used of data abstractions.
3. Each use of a data abstraction in the system occupies a distinct place in the usage hierarchy.

The box structures of the method are three different ways of viewing parts of the system under development. Essentially a data abstraction is defined in three forms, using boxes. Beginning with a black box, the developer expands the description into a state box, and then into a clear box. With each expansion, there is an immediate verification step that ensures the correctness of the expansion, or transformation.

More specifically, the black box view provides a description of the external behavior of a defined data abstraction in terms of stimulus-response. The initiation of a black box with an entry and any required data is described as the stimulus, and the completion of the black box, with an exit and the data that may be produced, is described as the response. This stimulus- response description is seen as a mathematical function from the stimulus history of the black box to the next response of the black box.

Expansion of the black box into a state box requires that the stimulus history of the black box be transformed into a state description. This is described as a mathematical function where the stimulus and state histories correspond to the new response and state.

The clear box view describes the behavior in terms of the four procedural structures needed to derive the state and black box views. The structures are sequence, alternation, iteration, and concurrency. These structures are used to replace the internal data abstraction defined in the state box.

Assisting in the effective use of box structures for development of information systems are four principles: referential transparency, transaction closure, state migration, and common services.

3.4.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

The developer rated the method well-suited for all applications. Examples of delivered systems developed using the method are large-scale simulation, forms processing, and satellite tracking. Box Structures have been used on less than five delivered systems; between five and twenty organizations have used the method. The method is intended for and has been used on projects of all sizes. Most frequently used implementation languages on projects developed with the method have been COBOL, C, and Pascal.

Target Constraints

The method can handle a number of target constraints, including timing and spatial constraints, special features of the target hardware architecture and operating system, concurrency and fault-tolerance issues, and security of access. These target system requirements are stored in box structure formats.

Furthermore, if a designated target system is not a requirement, then box structure designs are independent of target hardware and software.

Modes of Expression (Tables 9,10)

Textual modes of expression which are required and provided with automated support are structured English, a program design language, formal specification languages, and mathematical notation. Hierarchy charts is a required iconographical mode. Finite-state diagrams and entity-relationship diagrams are strongly encouraged. All of these iconographical modes are provided with automated support.

Creative invention is required to go from a black box to a state box, and from a state box to a clear box. However, analysis and mathematical derivation techniques allow one to go from a clear box to the corresponding state box, and from a state box to the corresponding black box. Other mappings are not required, as all representations are in box structures. These box structures are the integrating concept across the complete system (and software) development process.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

A number of techniques are strongly encouraged by the method for clarifying system requirements or behavior. Required for analysis and review are formal proof techniques and design reviews.

Other Technical Aspects

Concerning the assistance provided to incorporate changes in the requirements, box structures represent requirements. Changes in the requirements necessitate modifications in the box structures. The method provides a principle called "referential transparency" in order to isolate these changes.

Since specification, design, and code are all stored in box structure formats, consistency and integrity checking are straightforward. The uses hierarchy of the box structure method provides a disciplined, rigorous means for performing object-oriented design, which is seen to assist in the identification of potential reusable components.

3.4.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

The method provides a framework for performing activities associated with project management.

Communication Channels

The developer stated that the underlying mathematics-based theory and representation allow precise technical communication and management control. Between the client and the development organization, box structure graphics provide a visual means of client communication. These box structures are used during analysis and design to review with the client the requirements which he has presented.

Quality Assurance (Tables 12,14,15)

The method requires test planning at precise points in the software process, and provides guidelines for the prescriptive checking of interfaces. The method requires a quality assurance or test plan document.

Specific directions are provided for maintaining a record of technical decision-making, including specification or design options, trade-off studies, rationale for any decision, personnel involved in making decisions, and changes related to specification or design changes.

Documentation Formats (Table 16)

All documents required to be produced are tailorable in format, and generated automatically based on data produced from other steps in the method. See the associated table for specific documents.

3.4.5 EASE OF USE

Technology Insertion

In the opinion of the developer, minimum qualifications needed by a development team leader for successful use of the method include a bachelor's degree, one to two years of development experience, working knowledge of two programming languages, and experience with three or four different software systems. In addition, concepts which should be understood by an experienced developer for successful use were software engineering concepts, object-oriented methods, as well as mathematical correctness of systems and programs.

Training is available in the form of hands-on demonstrations, overview presentations, classroom tutorials, on-site consulting, video tapes, related publications from third-parties, and periodic technical updates.

A project manager would require 15 days to acquire an understanding of the major features and benefits of the method, while an experienced developer would need 60 days to learn to use the essentials. Six months would be required for an experienced developer to achieve expert user level.

Automated Facilities

Provided with the method is automated support for the textual and iconographical modes used by the method and the documents required to be produced (see Tables 9, 10, and 16). Useful tools supporting the activities of the method would be a word processor, a database system and a report generator.

3.4.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

A PC, like PS/2 50, or a workstation, e.g., Sun, are appropriate for hosting the tools of the method.

Acquisition Costs

Cost to acquire the method, including hardware, software, and training, is $10,000. Technical training alone is $3,000, and a management overview is $1,000. There is no licensing policy.

Contact Information

Information Systems Institute 407-569-3722
2770 Indian River Blvd.
Vero Beach, Florida 32960 [Provider]

3.4.7 REFERENCES

[Hevn88] A. R. Hevner, "An Integrated Systems Development Environment with Box Structures", paper presented at INTEC Symposium on Systems Analysis and Design, Atlanta, Oct./Nov. 1988.

[Mill88] H. D. Mills, "Stepwise Refinement and Verification in Box-Structured Systems", IEEE Computer, Vol. 21, No. 6, June 1988, pp. 23-36.

[Mill87] H. D. Mills, R. C. Linger, and A. R. Hevner, "Box Structured Information Systems", IBM Systems Journal, Vo. 26, No. 4, 1987, pp. 395-413.

3.5 BYRON -- Byron PDL and Document Generator

3.5.1 BACKGROUND

Synopsis

The Byron Program Design Language (PDL) and Document Generator supports an approach to software development that uses an Ada-based PDL together with keyword-based comments as a basis for defining, capturing information on, and expressing high-level and detailed design. It produces documentation as an automated byproduct of this process. Byron supports the use of keyword-based comments to express system specifications that result from the use of PSL/PSA or other methods.

History

Byron has been available and marketed since 1982. First used in development of a deliverable system in 1983, it was originally built to generate Mil-Std Design Specifications (C5s) for the U. S. Air Force Ada Compilation System (ACS) and was written in Pascal. As of March, 1987 Byron accepts the full Ada language and is totally written in Ada.

3.5.2 DESCRIPTION

Byron, consisting of the Byron PDL and Document Generator, provides guidelines for requirements definition, system specification, system design, and software quality assurance and also prescribes specific directions and procedures for system implementation. It addresses preliminary design, detailed design, and coding activities, as well as providing for configuration management, documentation, integration, and maintenance. It provides a PDL processor, the Byron Analyzer, a Program Library Manager, and three documentation tools.

Byron is a program design language which is an extension of the Ada language. Byron is itself completely written in Ada and uses the front end of a validated Ada compiler, the Byron Analyzer, to ensure that the PDL can be successfully compiled. The Program Library Manager supports revision control and allows links to other users' libraries. Byron's Document Generator takes input from the Program Library and produces documents according to predefined and user-defined templates, including the reports specified by Mil-Std 2167.

A high-level design is expressed initially using the Byron PDL. It is then entered into the program library and maintained there. More detail is then added to the design until a fully implemented, compiled Ada program results. Byron may be used in conjunction with functional or data-oriented decomposition methods. It can be incorporated into an integrated environment or provide a framework for customizing the software development process with the use of other methods and tools.

Byron is frequently used in conjunction with PSL/PSA for requirements definition and system specification. Intermetrics is currently working on a prototype to interface directly between PSL/PSA and Byron. The RSI tool provided by MetaSystems for use with PSL/PSA may also be used to produce such an interface.

3.5.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

The developer states that the method is well-suited for use in the areas of embedded systems/process control, time critical/real-time systems, scientific/engineering, systems programming, and image processing/pattern recognition. It has been used in more than one hundred organizations for completing an estimated twenty-one to one hundred medium and large projects.

Since Byron is an extension of the Ada language, it possesses Ada's features of readability and modularity; it supports structured programming; design decisions can be deferred; and it supports data abstractions and information hiding. The developer states that it is also compatible with the concepts and practices of stepwise refinement and genericity.

Target Constraints

Byron can be used to capture information concerning target constraints through its pre-defined or user-defined keyword capabilities. Additional design information, not expressible or required in Ada, can be stated through the use of Byron constructs embedded in Ada comments. The choice of an Ada-based PDL is intended to reduce the effort required to port end-product systems to different target configurations and to adapt end-product systems to new applications.

Modes of Expression (Tables 9,10)

Byron utilizes textual, rather than iconographical, modes of representation. It provides an Ada-based program design language and automated support. It requires the use of narrative overviews of modules.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

Byron typically relies upon the use of PSL/PSA for requirements specification. Byron provides guidance for the transformation of the evolving software across phases of the software process. The phase is specified as a parameter to the tool. The later the phase of the software process, the more information required in the code and options file. The tool will give warnings if the source code is not consistent with the phase.

Other Technical Aspects

Because the Byron PDL information is placed within Ada comments, the annotated source can be processed by the compiler. Thus, code and commentary can be maintained together, ensuring that documents will always reflect the latest version of the program. The source code is maintained in the program library in an intermediate form (DIANA) where it is available for further development work as well as for use by the documentation tools.

3.5.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

Byron does not provide managerial assistance, but does provide technical capabilities that aid in project management and control, as described in the following sections.

Communication Channels

Byron facilitates and coordinates communication within the development team and with the client, during the software process and continuing development, maintenance, and support. The master library and annotated comment structure of Byron produce technical documentation, providing information about design to other members of the development team during the later phases of the development process, as well as during maintenance and support, when the original designers are not available for questions. The phase checking done by the Byron Analyzer and configuration management and revision control of the program library also assist in technical control and communication. The automatic document generation capability is designed to facilitate and coordinate communication between the client and the development organization.

Quality Assurance (Tables 12,14,15)

Byron provides support for assessing conformity of the developing software to system specifications by utilizing Ada's capabilities for prescriptive checking of interfaces, by its relationship with the PSL/PSA formal specification language, by its provisions for embedding requirements and design information as annotated comments within the code, and by its incorporation of tools leading to verification. The user specifies the appropriate degree of completeness checking, done in *conjunction* with Ada rules.

Byron provides early detection of inconsistencies and/or errors based on an unambiguous PDL and automated interface checking. The front end of the Ada compiler will check syntax and semantics of the code, detecting programming errors such as type and naming inconsistencies. The post-semantic processor will detect errors and missing information in the comments. Correct completion of the method is confirmed through "phase checking", which checks for required information as a function of the asserted degree of completion.

The Program Library Manager supports configuration management in that it:

- generates builds and releases of the software;
- tracks multiple versions or variants of software;
- can re-create any version or variant;
- insures that changes are made to the appropriate version;
- prevents multiple concurrent updates;
- provides tracing of recompilation dependencies;
- helps to determine what needs to be retested-after changes.

Documentation Formats (Table 16)

The Byron PDL package includes three documentation tools. The Document Generator requires the use of a document template. A user can create new templates, modify existing templates, or use those provided. The predefined templates include a call-tree generator, data dictionary, user manual, dependency table, Mil-Std C5, STLDD (Software Top Level Design Document), and SDDD (Software Detailed Design Document). The

Program Library Access Package allows the user to write Ada programs to access the contents of the program library. It is used in conjunction with the VAX Ada compiler. The third tool is a general purpose text formatter.

3.5.5 EASE OF USE

Technology Insertion

The developer states that Byron can be used successfully by a development team leader with two to three years of college-level technical education, less than one year of development experience, a working knowledge of one programming language, and development or maintenance experience on one software system. A project manager could acquire an understanding of the major features and benefits of Byron in one day, while an experienced developer could learn to use the essentials in five days and could become an expert user of the method in two months.

Training is available in the form of user manuals, on-line tutorial and help facilities, "hot-line" service, periodic technical updates, and on-site consulting by independent consultants. There is a users support group, the Byron Users Group, which meets twice a year at SIGAda/AdaJUG meetings.

Automated Facilities

The method embodied in Byron cannot be separated from the automated tool. It can be incorporated in an integrated environment or provide a framework for customizing the software development process with the use of other methods and tools.

The available CASE tools include several supplied with the method by Intermetrics. These are Byron PDL, Byron/Ada (the part needed for the Analyzer), Byron Analyzer, and the Byron Document Generator In addition, several tools are supplied by other vendors. These include PSL/PSA from Meta Systems, Xinotech Program Composer from Xinotech Research, Inc., Keyone from LPS, and Procap from Promod, Inc.

3.5.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

Byron requires a DEC VAX/VMS.

Acquisition Costs

The unit cost to acquire the me'hod and the required components is estimated at $8,000 to $35,000. The unit cost for technical training is between $5,000 and $9,000. Primary licenses, secondary licenses, site licenses, and corporate licenses are available for purchase.

BYRON

Contact Information

Intermetrics 733 Concord Avenue Cambridge, MA	617-661-1840 [vendor]

3.5.7 REFERENCES

Information is available from the developer.

3.6 CORE -- Controlled Requirements Expression

3.6.1 BACKGROUND

Synopsis

CORE is aimed at the requirements analysis phase of software development. It attempts to create a model of the required system whose behavior can be understood by customer and analyst alike, and to derive requirements of functions, performance, and other attributes in a complete and consistent manner. CORE emphasizes the importance of the different groups of people involved in the development effort by composing the system in terms of the different "viewpoints" of these groups as well as by assigning specific roles to representatives from these groups.

History

CORE was developed by Systems Designers in Camberley, England, and by British Aerospace in Warton, England. It was developed in the late 70s and shares techniques in common with other requirements and specification methods. It was first used for the development of a deliverable system in 1980.

3.6.2 DESCRIPTION

CORE, Controlled Requirements Expression, addresses the very earliest stage of concept identification or requirements elicitation. It involves requirements analysis, risk analysis, and system specification. The approaches upon which the method is founded are data flow, data structure, control and event-oriented approaches, as well as functional decomposition. It is well-suited for use in the context of the Waterfall, the spiral, and the rapid prototyping paradigms, and considers stepwise refinement an essential concept.

The method starts by identifying the organizations or individuals who have a "viewpoint" in the proposed system. The concept of viewpoints is fundamental to CORE, and adopts the premise that different groups see the system in different ways and prefer different formats for presenting information. These groups are formalized into the roles of Customer Authority, User Authorities, and Analysts. Each role has defined responsibilities and powers for arriving at a requirements specification acceptable to all parties.

This requirements specification is reached by means of the following seven stages in the method:

1) Problem Definition,
2) Viewpoint Structuring,
3) Tabular Collection,
4) Data Structuring,
5) Single Viewpoint Modelling,
6) Combined Viewpoint Modelling,
7) Constraints Analysis.

In the first stage, problem definition, the business or strategic objectives of the system are documented from a senior management perspective. This perspective would include a statement of the current problems with the existing system (if one exists), goals to be achieved by the new system, indications of future plans or future directions for the system, and initial constraints. At this stage, the Customer Authority is identified.

In the next stage, viewpoint structuring, relevant views are drawn up to represent the system and its environment. These viewpoints may be derived by considering relationships like supply and support of the system, e.g., design or maintenance; users of the system, e.g., companies or organizations; operational aspects, e.g., embedded systems; and functional aspects, e.g., logical groups of functions. The Customer Authority is involved at this stage to determine which aspects require analysis to become direct viewpoints of the system, and which aspects are sources or sinks of information and can be considered indirect viewpoints. Viewpoint Authorities are selected, each to be a spokesperson for a particular direct viewpoint. Also at this stage the Analysts can provide an assessment of the work required for the analysis task.

Stages 3 through 7 of the method are performed iteratively to successively refine each viewpoint level. Stage 3, tabular collection, is concerned with collecting, representing, and checking information provided for each direct viewpoint in the viewpoint hierarchy. This information is derived for a particular viewpoint by interviewing the Viewpoint Authority or by examining relevant documents. The information is presented in a five-field form, showing sources, inputs, actions, outputs, and destinations. The fields are connected by arrows to show the flow and transformation of information within the viewpoint. Data structuring then is carried out, examining the data in terms of its content, order, grouping, and repetition. Single viewpoint modelling takes the information collected in the tabular collection and data structuring stages and creates a diagram to show the inputs, outputs, and action of a viewpoint. These Single Viewpoint Models show the data-flows relating to other viewpoints and the internal dataflows that link up the actions within the viewpoint. Next, a similar diagram, called a Combined Viewpoint Model, is constructed in order to either confirm that the children viewpoints fulfill the definition of their parent viewpoint, or to represent cross-system transactions involving multiple viewpoints. Finally, constraints analysis is applied to the model of the system.

3.6.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

The developer rated the method well-suited for applications areas such as embedded systems or process control, time critical or real-time processing, distributed processing or networks, large-scale simulation or modeling, and analysis of human/non-data processing systems. Delivered systems built with the method include avionics software, naval systems, gas industry systems, and space station software. Between 21-100 delivered systems have been developed using CORE, within 21-50 organizations. The method is intended for use on projects of all sizes and has been used as such. Implementation language is not really relevant in the developer's opinion; however, the implementation languages most frequently used with the method were CORAL66, Ada, and Pascal.

Target Constraints

The method addresses concurrency issues by means of the Single Viewpoint Diagrams, in which critical and non-critical dataflows can be used to identify concurrency. Fault tolerance issues are addressed via the Combined Viewpoint modeling, where specific fault tolerance situations can be identified and studied.

Modes of Expression (Tables 9,10)

Data-flow and control-flow diagrams are required; specified documentation templates, narrative overviews of modules, and structured English are strongly encouraged. The method's Tabular Collection diagrams identify inter-viewpoint dataflows and actions. These can be used to provide a starting point for Single Viewpoint models, thus assisting in translating from one mode of expression to another.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

Required techniques are data-structure, data-flow, and control-flow analyses. Dynamic animation and simulation are possible and encouraged for clarification of system requirements.

Other Technical Aspects

As a method which specifically addresses requirements identification, CORE supports the identification of changes to the requirements. There are built-in self-consistency checks designed to identify inconsistencies and errors. CORE can also be used retrospectively to analyze existing system specifications.

3.6.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

Specific directions are provided for analyzing risk; other project management activities are not addressed specifically by the method.

Communication Channels

Diagrammatic representations are designed to give good visibility of the requirements between all parties involved in the creation of a software system. The client is involved in the software development process since the method requires nominating a "spokesperson" to represent each group of organizations or individuals associated with the application area. These groups form the "viewpoints" of the system.

The method prescribes the responsibilities of these spokespersons according to the roles they will take. The Customer Authority role involves defining the scope and plan for analysis, identifying the User Authorities (later called Viewpoint Authorities), resolving disputes, and deciding on all changes of scope and planning. User Authorities are the sources of information for the Analysts, and must provide a description of the system in terms of any predecessor, as well as the interface desired. The Users must also come to agreement among themselves on the description. The role of the Analyst is to conduct interviews with the Customer and Users and structure the specification. In addition, the Analyst must make the specification understood by all parties and check the specification for inconsistencies, omissions, and premature design. Finally, the Analyst has responsibility to see that conflicts are resolved by the Customer, that decisions are recorded and understood, and that no information is lost.

Quality Assurance (Tables 12,14,15)

CORE does not address this aspect; it only addresses requirements identification.

Documentation Formats (Table 16)

The method does not prescribe document formats although it does prescribe specific types of representations.

3.6.5 EASE OF USE

Technology Insertion

In order to successfully use the method, a development team leader need have less than two years of college-level technical education, one to two years of development experience, and experience with at least two software systems. No programming language experience is necessary. In addition, use of regular grammars, data-flow techniques, and control-flow techniques should be understood, but are not required as a pre-condition since they are all covered in the standard CORE training course.

Training assistance is available in the form of hands-on demonstrations, overview presentations, classroom tutorials, and on-site consulting by the vendor. There also are on-line help facilities, a "hot line" service, and user manuals. A project manager would need one day to acquire an understanding of the major features of the method, while an experienced developer would need five days to learn to use the essentials of the method, and six months to become an expert user.

Automated Facilities

The developer provides a tool called "Analyst", which assists in the production and checking of CORE diagrams and provides training support.

3.6.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

A MacIntosh2 is appropriate for hosting the method's tool.

Acquisition Costs

Cost to acquire the method for a single user is $5,000. F · low-volume users it is $3,000, and for high-volume users, $2,000. Technical training is $1,500 and a management overview is $300. There is a licensing policy.

Contact Information

SD-Scicon PLC 0276 686 200
Pembroke House, Pembroke Broadway
Camberley, Surrey GU15 3XD, England [provider]

3.6.7 REFERENCES

[Mull82] G. Mullery, "CORE, A Method for Controlled Requirements Specification", Proceedings of the 4th International Conference on Software Engineering, sponsored by the IEEE, 1982.

3.7 DARTS -- Design Approach for Real-Time Systems

3.7.1 BACKGROUND

Synopsis

This method extends Structured Analysis/Design into the real-time problem domain by providing design criteria for multitasking. The steps of the method provide structural criteria for identifying concurrent tasks and guidelines for defining task interfaces. These steps occur after a structured system specification has been developed and before each task is designed. The method is available for general use.

History

The precursors of DARTS are Real-Time Structured Analysis and Structured Design. Dr. Hassan Gomaa developed the method in order to take into account the characterisitcs of real time systems which typically consist of several concurrent tasks or processes. The method was first used for the development of a deliverable system in 1982.

3.7.2 DESCRIPTION

DARTS, Design Approach for Real-Time Systems, addresses the activities of software system specification and design. The method is founded upon approaches which are data flow-oriented, control-oriented, and event-oriented. A number of software process paradigms are considered to be well-suited for use with the method, although the most effective use is not dependent upon any particular paradigm. However, the method does encourage incremental development or evolutionary prototyping, and considers the transformational model and the 4GL model inappropriate for use with DARTS. Essential concepts of the method include information hiding, abstract data-types, structured programming, and module coupling/cohesion.

DARTS may be considered an extension of the Real Time Structured Analysis and Structured Design methods by providing an approach for structuring a real time system into concurrent tasks as well as a mechanism for defining the interfaces between tasks. The steps of the method are:

1. Develop Structured System Specfrication using Real Time Structured Analysis. The Ward/Mellor, Boeing/Hatley or ESML approaches may be used.
2. Structure the system into concurrent tasks using the task structuring criteria.
3. Define task interfaces.
4. Design each task, which represents a sequential program, using the Structured Design method.

The task structuring criteria guide the designer in decomposing a real time system into concurrent tasks. The main consideration in identifying tasks is the asynchronous nature of the functions within the system. A task may exhibit more than one of the task structuring criteria. The Event Dependency criteria include Device I/O Dependency, User Interface Dependency, Periodic, Periodic I/O and Entity Modelling. The task cohesion criteria include Sequential Cohesion, Temporal Cohesion, and Functional Cohesion. The Task priority criteria are the Time Critical and Computationally Intensive criteria.

In defining Task Interfaces, a data flow between two tasks is designed as a message. Either loosely coupled or tightly coupled message communication is supported. Event signals are used for synchronizing purposes between two tasks when no data needs to be communicated. Data stores are encapsulated into information hiding modules, in which the data structure is defined as well as the access procedures. The access procedures also synchronize access to the data.

3.7.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

The developer states that the method is well-suited for applications involving embedded systems or process control, time critical or real-time processing, distributed processing or networks, and image processing or pattern recognition. Examples of delivered systems developed with the method are a robot controller and an image processing and pattern recognition system. The method has been used by between five and twenty organizations, with between five and twenty delivered systems developed with the method. The method has been used on small and medium-sized projects; it is intended for projects of all sizes. Most frequently used implementation languages for systems developed with the method are Pascal and C.

Target Constraints

DARTS prescribes steps for handling several requirements of the target system. Central to the method are concurrency issues. The method also addresses timing constraints through the use of event sequence diagrams which trace critical external events through the system. Special features of the target hardware architecture are handled by allocation of tasks to CPUs. Message communication and event synchronization for task interfaces are the ways the method handles special features of the target operating system. Finally, the design documentation of the method is seen by the developer as assisting in porting end-product systems to different target configurations.

Modes of Expression (Tables 9,10)

The method requires structured English as a textual mode of representation. Required iconographical modes include finite-state diagrams, data-flow diagrams, control-flow diagrams, and Structured Design structure charts. The method prescribes mapping rules for translating from data-flow diagrams to task structure diagrams, and from task structure diagrams to module hierarchies. The developer states that transformation across phases of the software process is facilitated by the criteria and guidelines for performing transformations.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

A number of techniques for clarifying system requirements are encouraged by the method. Data-flow and control-flow analyses are required, as well as design reviews.

Other Technical Aspects

By emphasizing task structuring and information hiding, the method is reported to facilitate incorporation of changes. By requiring design reviews, the method is intended to ensure consistency between specification, design or code. Information hiding modules is also seen to assist in identification of possible reusable design components.

3.7.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

DARTS does not address project management issues.

Communication Channels

The specific features of the method designed to facilitate communication within the development team are the iconographical modes of representation used and the decomposition of the system into tasks or modules with well- defined interfaces. Between the technical development team and management, communication is facilitated by means of an incremental development approach and well-defined milestones and deliverables. Between the client and the development organization, the iconographical modes of representation are designed to facilitate communication. In addition, the client is involved in the development process by means of reviews during specification, and by means of rapid prototyping of the user interface.

Quality Assurance (Tables 12,14,15)

DARTS requires testing activities and recording of technical decision making during the software development process, but does not provide directions for accomplishing these activities. It assists in the early detection of inconsistencies and/or errors by means of design reviews, rapid prototyping, and incremental development.

Documentation Formats (Table 16)

Several documents are required to be produced by the method. Where the method prescribes the format, the format is either fixed or tailorable.

3.7.5 EASE OF USE

Technology Insertion

The developer estimated the following minimum qualifications needed by a development team leader for successful use of the method: a bachelor's degree, three to five years of development experience, working knowledge of two programming languages, and experience on two different software systems. Major theoretical concepts which should be understood are finite-state machines and concurrent tasks.

The developer offers overview presentations, classroom tutorials and on- site consulting as training in the method. It was estimated that a project manager would need one-half day to understand the major features of DARTS, while experienced developers would need one day to learn to use the method's essentials and 3 to 6 months to achieve the level of expert user.

Automated Facilities

Not applicable.

3.7.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

The method is compatible with SA/SD PC tools and workstation tools.

Acquisition Costs

Not applicable.

Contact Information

Dr. Hassan Gomaa 703-764-6191
George Mason University 703-323-3530
School of Information Technology & Engineering
4400 University Drive
Fairfax, Virginia 22030-4444 [Provider of method]

3.7.7 REFERENCES

[Goma84] H. Gomaa, "A Software Design Method for Real Time Systems", Communications of the ACM, Sept. 1984.

[Goma86] H. Gomaa, "Software Development of Real Time Systems", Communications of the ACM, Vol. 29, July 1986, pp. 657-668.

[Goma87] H. Gomaa, "Using the DARTS Software Design Method for Real Time Systems", Proceedings of the Twelfth Structured Methods Conference, Chicago, Aug. 1987.

[Goma88] H. Gomaa, "Extending the DARTS Software Design Method to Distributed Real Time Applications", Proceedings of the 21st Hawaii International Conference on System Sciences, Jan. 1988.

3.8 **DBO** – Design by Objectives

3.8.1 **BACKGROUND**

Synopsis

Design by Objectives requires the identification of system objectives and the assignment of quantitative measures to all system attributes. Such measures allow these attributes to be integrated with each other and tested, providing information for the trade-off analysis of different technical solutions so as to determine whether objectives are being met. The method addresses quality control and advocates the incremental development of well-understood subsets of the system for early validation.

History

DBO was mainly developed by Tom Gilb. Dr. Lech Krzanik developed the Aspect Engine software tool for use with DBO; the Inspection part of the method originated with Michael Fagan. The basic ideas evolved over the decade spanning the 1970's and have been elaborated in a textbook by the developer. Components of the method were first used with respect to a deliverable system in 1968 and earlier.

3.8.2 **DESCRIPTION**

Design by Objectives (DBO) incorporates a number of techniques for problem and requirements definition, specification, analysis, estimation, preliminary design, planning, testing and maintenance. It views the construction of software as but one part of a total system.

The following passage from [Pete81] explains a central concept of DBO, which is concerned with how attributes shall be specified:

> "The emphasis in DBO is on what is measurable, rather than what is desired. For example, if the customer wishes to have a system developed to improve customer service, this must be stated in a measurable way, in terms of an index or some such measure of customer service whose minimum value and desired value after installation have also been established."

The basic method consists of the following steps:

- Decide initial functional requirements, using any method;
- Decide initial attribute (quality and resource) requirements which are quantitative and testable;
- Identify initial solutions for meeting attribute requirements;
- Estimate effects of all solutions on all attributes;
- Make and execute an evolutionary step-wise system delivery plan for early experience and feedback;
- Use Fagan's Inspection method on all written material;
- Iterate until all requirements are satisfied.

The attribute specification for the system includes such items as definition of scales of measure, ways of testing the attribute, time or condition when the measure is applicable, worst acceptable case level, and planned levels of acceptability. Attributes may also be defined with a set of sub-attributes. The functional solution and delivery step specifications consist of hierarchical lists of ideas which can be "tagged" for quality control crosschecking.

The method requires that specifications be analyzed in order to understand the consequences of multiple technical design decisions. The impact of changes to one attribute is assessed across other attributes by means of impact estimation tables and impact analysis tables. Solution comparison tables are used to determine which technical solution has the best net impact on a set of attribute level requirements. The method also makes use of an "engineering handbook", which is a database of potential technical solutions containing information on the degree to which, according to certain criteria, a particular solution is expected to measurably affect a selected set of attributes. Early quality control is provided for all DBO documentation by the use of Fagan's Inspection Process.

3.8.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

The developer regards the method as independent of the type of application in the sense that DBO can also be used for the development of non-computer systems. Examples of applications for which delivered systems have been built using the method include computer supplier software, aircraft co-simulations, bank/insurance applications, software organizational planning, and software package selection. The method is considered to be independent of size and language as well; projects of all sizes have been built with DBO and a variety of implementation languages have been used with it. The method is employed at the architectural and design level, not at a program logic level.

In summary, more than 250 delivered systems have been developed using the method within more than 100 organizations. The developer stated that these figures include some applications of usage without all components or tools associated with the method.

Target Constraints

The developer reported that DBO prescribes steps for handling timing and spatial constraints, as well as special features of the target hardware architecture and operating system; further description of how these issues are addressed by the method was not provided. Concurrency and fault-tolerance issues, and security of access are not explicitly addressed, but the developer considers that these are issues that could be addressed along with any other design issue.

Because DBO is not a programming method, it does not deal specifically with portability issues; however, attribute specification is strongly encouraged to specify the degree and types of portability, which are then engineered into the system.

Modes of Expression (Tables 9,10)

The developer stresses the fact that DBO operates at a level above the detail level where most textual and iconographical modes of representation are used. Therefore, the modes of representation shown in Tables 9 and 10 are considered to be largely outside the scope of the method.

DBO's textual modes of representation include impact estimation, a formally defined planning language called "Planguage", tagging and cross-referencing, and formal separation of objectives/solutions/delivery steps.

Decision tables are compatible with the method. Iconographical modes include N-squared charts, which are strongly encouraged.

Transformation across phases of the software process is accomplished by a) by relating design to the requirements specification quantitatively in all quality and cost dimensions, b) by Fagan's Inspection to ensure consistency after the transformation, c) by referencing "tagged" once-only definitions, d) by optional use of a CASE tool, "Aspect Engine", and e) by a relatively formal specification language, "Planguage".

Techniques for Analysis and Requirements Clarification (Tables 11,12)

The method requires the use of incremental or evolutionary development to clarify system requiements. The use of rapid prototyping is strongly encouraged.

Analysis and review techniques utilized are Fagan's Inspections (which replace design reviews and code walkthroughs), impact estimation of solutions on quality/cost/function, and measurement of properties at each evolutionary step delivery. Change Control Board review is also strongly encouraged.

Other Technical Aspects

The method assists in reducing the effort needed to fully incorporate changes in the requirements by an extensive network of "tags"; cross-references to these tags are used in all aspects of design and planning documentation. Also used is a disciplined application of Fagan's Inspection of all design documents and code. Thirdly, small-risk incremental build and change, or evolutionary development, is followed. These three aspects of the method are seen to assist in ensuring consistency of the developing system and the early detection of errors in the system.

3.8.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

DBO prescribes specific directions and procedures for conducting a variety of activities associated with project management, some of which are supported with automated tools. These are shown in Table 13.

In addition to the activities shown in the table, DBO has specific procedures for addressing issues concerning quality, in particular, tracking and estimating quality. The method does not address the assessment of complexity, as this is regarded as an intermediary device for assessing risk and cost, both of which the method addresses specifically.

Communication Channels

Particular to the method's facilitation of communication within the development team is the quantified definition of all critical quality/advantage/resource results. All other sub-techniques relate to this aspect.

In addition to the above, several features of the method are designed to facilitate and coordinate communication between management and the technical development team, and between the client and the development organization. These include attribute specification, impact estimation, inspection, and evolutionary delivery. The method requires substantial involvement of the client in the software development process. The initial involvement includes the negotiation of attribute specification, which is essentially a contract for results. Additionally, the client participates in determining process standards and in inspecting documents. He or she may also be involved in impact estimation reviews. Finally, the client is involved when evolutionary steps are delivered either to the client or to a simulated use environment.

Quality Assurance (Tables 12,14,15)

DBO specifically addresses test planning, generation of tests based on system requirements, and early "testing" of documentation using Fagan's Inspection. Other verification activities are outside the direct scope of the method.

Automated recording procedures and specific directions are provided by the method for maintaining records of technical decision-making: the specification/design options considered, trade-off studies, rationale for any decisions, personnel involved in making decisions, and all changes related to specification/design decisions.

Documentation Formats (Table 16)

All of the documents required to be produced by the method do not always require use of automated facilities. However, several of them can be produced with the "Aspect Engine", a prototype tool. Normally other support tools are used (e.g., PC word processors, outlines, spread sheets) For formats and type of automated support for these documents, see Table 16.

In addition to the above, the method requires impact estimation tables, evolutionary delivery step specification, inspection process standards, and inspection checklists. All of these documents receive automated support or can be manually produced.

3.8.5 EASE OF USE

Technology Insertion

The developer estimated that a team leader would need less than two years of college-level technical education and three to five years of development experience. Knowledge of programming languages and experience on different software systems were not considered necessary prerequisites for a team leader.

Training includes hands-on demonstrations, overview presentations, classroom tutorials, on-site consulting, video tapes, an on-line help facility, user manuals, related publications from third-parties, and periodic technical updates. One day is required for a project manager to acquire an understanding of the major features of DBO. A minimum of five days would be required for an experienced developer to learn to use the essentials of the method, five to ten days for subcomponents like "Inspection". About one to two months would be required for a developer to achieve expert user level.

Automated Facilities

A prototype tool, the Aspect Engine, supports all of the activities of the method. See Tables 13 and 16. The developer mentions that normally other support tools are used for the method's activities, including PC word processors, outlines, spread sheets, and charting facilities.

3.8.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

An Apple MacIntosh is appropriate for hosting tools associated with the method.

Acquisition Costs

Technical training for low-volume users is $2000, and for high-volume users is $500. A description of the method is available in the book Principles of Software Engineering Management by Tom Gilb.

Contact Information

Absolute Software
4593 Orchid Drive
Los Angeles, CA 90043

213-293-0783

[Provider of method]

FI-LA Marketing
14 Junction Rd.
London W54XL
England

01-847-0471

[Provider of tool,
manuscripts]

3.8.7 REFERENCES

[Faga76] M. E. Fagan, "Design and Code Inspections to Reduce Errors in Program Development", IBM Systems Journal, Vol. 15, No. 3, 1976, pp. 182-211.

[Gilb88] T. Gilb, Principles of Software Engineering Management. Addison-Wesley, 1988.

FI-LA (see Contact Information) provides a 450 page manuscript by T. Gilb, "Software Engineering Design". They also provide a working prototype of the Aspect Engine for the MacIntosh, as well as a wide variety of teaching materials and articles on MacIntosh disks by Gilb.

3.9 **DCDS** -- Distributed Computing Design System

3.9.1 BACKGROUND

Synopsis

DCDS provides an integrated approach for supporting the representation of system and software requirements. It also supports the design and testing of systems. It provides languages supporting each phase of development, with an emphasis on traceability between phases. A common language syntax is used at all levels; common tools are used to translate the languages into a data base, analyze them for consistency and completeness, and extract information for specifications and reports.

The method is available for general use, upon authorization from USASDC. The Systems Engineering phase of DCDS has been productized with the RDD-100 tool commercially available from Ascent Logic Corporation.

History

DCDS was developed by Mack Alford, Robert Loshbough, and others at TRW since 1973. In 1979 DCDS was first used on FAS/ADOP. From 1980-1984 it was used to develop a project which provided real-time distributed processing of optical tracking data. Earlier versions of DCDS are known as SREM (Software Requirements Engineering Methodology) and REVS (Requirements Engineering Validation System).

3.9.2 DESCRIPTION

DCDS, Distributed Computing Design System, is a collection of procedures and tools for addressing various stages of development. These procedures include the use of the System Requirements Engineering Methodology, the Software Requirements Engineering Methodology, the Distributed Design Methodology, the Module Design Methodology, and the Test Support Methodology. DCDS is founded upon several approaches to software construction and is said to be well-suited to a number of software process paradigms. Specifically, the componert methodologies of DCDS include risk management and guided flexibility in the choice of a path through the steps, based on Boehm's Spiral Model.

DCDS begins with the definition of system level requirements in a System Specification Language (SSL). SSL is used to define functions, their inputs and outputs, and their decomposition and allocations with respect to hardware components, including the data processor. SSL also defines the interface designs, and control functions dealing with resource management and fault tolerance. It is designed to preserve, by means of a graphic method, the previous specification of function sequences and concurrencies, the conditions for state transitions, and inputs and outputs. This phase of the method extends the MIL-STD-499 Functional Sequence Diagrams to allow specification of the system behavior with regard to each of the objects handled by the system. A simulation generator is available to generate a stand-alone Ada simulation of the system.

At the next level of the method, Software Requirements Engineering Methodology (SREM) is used to decompose the previous specification of functions to the state machine stimulus-response level of requirements. The Requirements Statement Language (RSL) used here overlaps with the SSL to preserve the specified functional behavior and insure that no data is used before it has been given a value, thus verifying the consistency of data flow. A simulation generator is available to generate an Ada simulation of the stimulus-response processing requirements.

The Distributed Design Methodology provides the language and tools for packaging the implementation processing on distributed processors. This step identifies the required concurrent units of code and their scheduling to satisfy response time requirements while at the same time satisfying execution time and memory constraints for each processor. This step also identifies ways to partition state information into data objects. The aim is to provide a way for deriving distributed real-time designs which preserve both stimulus-response and data flow requirements. A process construction tool accesses the design database to build source code for each processor.

The Module Design Methodology is then used to identify the application oriented modules, establish interfaces at the variable data-type level, and develop the modules using an Automated Unit Development Folder approach consistent with MIL-STD-2167. This step attempts to ensure strict traceability of the modules and data flow to the stimulus-response level of requirements.

The Test Support Methodology provides a language, tools, and procedures for developing the sequence of integration tests. These tests are intended to incrementally validate the code against the requirements as it is developed.

3.9.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

DCDS is reported to be well-suited for use on applications involving timing constraints and process control, among others. Specifically, the method has been used to develop the following delivered systems: OTH-B, TDRSS, FAS/ADOP (an embedded real-time application), BM/C3 CD, GSS, SSDS, EV88.

In sum, between 5 and 20 systems have been delivered using this method, and between 21 and 50 organizations have used the method. DCDS is intended for large projects, and has been used only on large projects.

While at present DCDS is language-independent, the Distributed Design and Module Development phases may be tailored to be Ada-specific by extending the underlying elements, relations, and attributes used to record design information. Pascal and Ada have been most frequently used for coding systems developed with the method.

Ascent Logic's RDD-100 is being used by GE to control the Phase I SDI system requirements and is in use at another five locations.

Target Constraints

There are a number of constraints which are addressed by language constructs within DCDS. General constraints are represented with "CONSTRAINT", with traceability to design objects. Timing constraints can be indicated at the requirements level by specifying COMPLETION_CONDITION (which is an attribute of FUNCTION), and "DATA name DELAYS EVENT name". In the design, XQT_BUDGET (an attribute of ROUTINE) can be specified for timing as well as SIZE (an attribute of ROUTINE) for spatial constraints. The special features of the target hardware and operating system can be addressed within the language structures for hardware design, including distributed design, message passing, event and state monitoring, priorities, call, and services. Methodology guidance is given for using these constructs, and for addressing concurrency and fault-tolerance issues using other language constructs. An analytical tool can be applied to concurrency problems ("CLUSTER data elements").

Guidelines for estimating timing and sizing of the target hardware and the method's distributed process construction technique are designed to reduce portability problems. Tasks and data are developed so that their locations are transparent to each other. Their allocation to processors resides in the database, which creates tables for identifying their locations. Tools can then be used to extract the appropriate code segments with respect to each processor. Simulations generated from the design specifications are used to verify response time satisfaction.

Modes of Expression (Tables 9,10)

DCDS textual modes include specified documentation templates, narrative overviews of modules, and formal specification languages. The required iconographical modes include control-flow diagrams and flowcharts. Automated facilities are provided for these required modes as well as for a number of other modes encouraged by the method.

DCDS prescribes mapping rules for translating between several of its modes of expression. In particular, F_NET (control flow) and text database information can be mapped into an IDEFo diagram (data flow). Logic flow diagrams are mapped into PDL, while extended entity-relationship-attribute (ERA) graphics translate into HIPO and Hierarchy charts. All the above mappings are automated by both DCDS and RDD-100.

Transformation of information across the phases of the software process is semi-automated. Upon completion of the system requirements database, the elements to be forwarded are automatically written to a file. When the software requirements database is opened, this file is input to provide the initial information. Then the elements from upstream must be transformed into elements of the downstream language. The same process is repeated when opening the distributed design phase, the module development phase, and the test phase. The languages for each phase are similar, though specific to the phase.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

DCDS requires simulation and executable specifications in order to clarify system requirements/behavior. Review techniques that are required include analysis of data structures, data flow, and control flow, as well as design reviews and code walkthroughs. "DECISIONS" are used to capture design rationale during development.

Other Technical Aspects

In order to facilitate changes in the requirements, DCDS uses two element types: DECISION can be used for special topics or refinements that arise during development, and CHANGE_REQUEST is used for indicating formal changes in baselined requirements. Additionally, DCDS has rules and ERA constructs to maintain traceability from end to end: requirements to design to code; and requirements to design to tests.

Regarding assistance for reusing components, the module development methodology gives guidelines for selecting modules to be kept in a reusability library. It also leads the user in setting up and maintaining a reuseability library on the DCDS tools. Reusability is also addressed by changing the desired behavior and seeing which functions remain. The remaining functions will map to the same modules. Since the module design is separate from the modules themselves, the modules can be reused.

Both DCDS and RDD-100 provide users with the capability of tailoring the elements, relations, and attributes of the database to a project, forming user-specified consistency checks on that information, and generating reports containing that information.

3.9.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

Specific directions and procedures are provided for analyzing risk, tracking project progress, project planning, scheduling and/or manpower loading, and configuration management. Automated support is provided for the above activities in DCDS.

Communication Channels

English-like ERA languages, extensive graphics, and automated document generation are designed to facilitate communication among all parties involved in a software development project. These can be extended and tailored to a project.

In addition, within the development team, there is a query language with the following features: retrieval of sets by relationship, attribute, qualifier, and hierarchy qualification; combination of information (AND, OR, MINUS); creation of hierarchies, allowing attributes and qualifiers as well as relationships, with various options to format listings; and a report command for formatting templates.

Project management is provided with detailed steps for determining quantified milestones, and given tailored methodology paths in the implementation of Boehm's Spiral Model for risk management.

For involving the software client in the software process, DCDS automatically produces a SSS for System Requirements Review, a SSDD for System Design Review, a SRS for Software Specification Review, a SDD (preliminary version) for Preliminary Design Review, and a SDD (detailed version) for Critical Design Review. DCDS also supports production of test reports for Functional and Physical Configuration Audits. RDD-100 currently supports only the Systems Engineering phase, generating a standard SSS, Interface and Traceability reports, as well as allowing the user to construct project-unique reports.

Quality Assurance (Tables 12,14,15)

DCDS and RDD-100 provide automated checking for completeness and consistency. A change would be incorporated by locating the first place where behavior changes, using automated tools to identify affected elements for an impact analysis, then modifying behavior, and using the tools to again assure consistency. A check is provided for use at the end of every major step.

Technical decision-making records are tailorable and maintained for automatic traceability of the progress of a project, in particular, logs on design decisions, problems, and changes. The method prescribes specific directions and procedures for configuration management, test planning and test generation.

RDD-100 is hosted on Sun 3/60, Apollo, and MacIntosh II having 8 MB memory. There are plans to host the tool on IBM PS/2.

Acquisition Costs

For DCDS, there is no separate cost for the method and software tools, other than the hardware and operating system configurations required for use of the tools. There is also no licensing policy.

Contact the vendor for cost information on RDD-100.

Contact Information

U.S. Army Strategic Defense Command 205-895-3858
Dr. Virginia Kobler (DASD-H-SBT)
P. O. Box 1500
Huntsville, Alabama 35807 [DCDS provider]

Ascent Logic Corporation 408-943-0630
Suite 200, 180 Rose Orchard Way
San Jose, California 95037 [RDD-100 vendor]

3.9.7 REFERENCES

[A'fo88] M. Alford, "RDD Approach to Systems Engineering", Requirements Driven Developer, Vol. 1, No. 1, Dec. 1988.

[Alfo87a] M. Alford, "DCDS Multiple View Approach to Closing the Requirements/Design Gap", presentation at the Fourth Conference on Methodologies aud Tools for Real-Time Systems, Washington, DC, Sept. 14-15, 1987.

[Alfo87b] M. Alford, "Requirements Driven Test Planning", presentation at Software Testing and Validation Conference, Washington, DC, Sept. 23-24, 1987.

[Alfo85] M. Alford, "SREM at the Age of Eight: the Distributed Computing Design System", Computer, Vol. 18, April 1985, pp. 34-36.

[TRW 87a] TRW System Development Division, Distributed Comupting Design System: "A Technical Overview". Huntsville, AL, 25 July 1987.

[TRW 87b] TRW System Development Division, Distributed Comupting Design System: "Tools User's Guide with Appendices". Huntsville, AL, Oct. 1987.

[TRW 87c] TRW System Development Division, Distributed Comupting Design System: "Methodology Guide with Appendices". Huntsville, AL, Oct. 1987.

Documentation Formats (Table 16)

DCDS provides a variety of documents, generally tailorable within the method and generated automatically as result of data produced from other steps in the method. The list of specific documents is given in Table 16.

RDD-100 provides a number of standard reports, and allows the user to generate his own reports.

3.9.5 EASE OF USE

Technology Insertion

Minimum qualifications for a software development team leader were estimated as a bachelor's degree, 3 to 5 years' development experience, working knowledge of 2 programming languages, and experience with 3 to 4 different software systems. In order to use the method successfully, an experienced developer would need to understand F_NETS (control flow structure) and R_NETS (stimulus-response thread structure).

Among the types of training assistance available are overview presentations, classroom tutorials, on-site consulting and help facility, "hot line" service, user manuals, and periodic technical updates. Estimates of learning times were 3 days for project managers, 5 days for experienced developers to understand the essentials of the method, and 2 months for an experienced developer to become an expert user of DCDS.

The developer of RDD-100 felt that learning time would be reduced with the tool because of the multi-window, menu-driven user interface.

Automated Facilities

DCDS includes a full suite of tools that are designed to support large projects. The method provides automated facilities for representations required by the method, as well as for other representations which are encouraged. The method generates a number of software specification and other documents. There are specific procedures for test planning, supported by automatic facilities, and guidelines for other types of test activities which are also automated. Automated procedures are used to maintain a number of records of technical decision-making during the development process, and project management activities specifically addressed have automated support.

Multi-user support is provided by DCDS in terms of an ASCII file export/import capability for sharing data among several users. RDD-100 supports element ownership with export/import. Thus, a master database can export a function and all of its related elements to an engineer for further refinement. The elements not "owned by" the engineer appear as "read only". All changes are transmitted back to the master database, where conflicts are automatically detected and diverted to a "conflict file" for case-by-case resolution.

3.9.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

DCDS tools may be hosted on a VAX with VMS 4.6, or on a SUN 3 system with Sun 05 3.4 or 3.5.

3.10 DSSAD -- Data Structured Systems Analysis and Design

3.10.1 BACKGROUND

Synopsis

DSSAD addresses activities associated with requirements definition or clarification, system specification, and system design. It is founded on a data structure-oriented approach and entity-relationship modeling.

History

The precursor of this method is Jackson Structured Programming. DSSAD's principal architect is Colin Knight. The method was first used for the development of a deliverable system in 1981.

3.10.2 DESCRIPTION

DSSAD, Data Structured Systems Analysis and Design, consists of nine steps, which, while themselves distinct, may overlap and reiterate upon each other. The developer rated the method well-suited for use in the context of the incremental model of the software process; he considers the method inappropriate for use with rapid prototyping.

The following paragraphs summarize the nine steps of the method:

1: Define the outputs. A system is bounded either by its inputs or outputs; a definition of either one defines the whole system. Since the user is interested in what the system produces, a definition of the output fixes the system. If the output is defined in terms of physical output forms, the abstract model will be revealed by removing the surrounding control and format data.

2: Identify the entities. Applying the techniques of data normalization reveals the entities underlying the abstract data model. Construct an entity/attribute list.

3: Construct entity behavior models. For each entity identified construct a behavior model for its total life from its beginning to the extinction of the entity. All the events that have any effect on the entity or are affected by the entity are part of the behavioral model.

4: Refine behavior models. Resolve any parallelism existing between event patterns that occur simultaneously. Refine the behaviors to account for the different types of relationship that can exist between event patterns.

5: Construct interaction diagrams. List all the entities affected by a particular event, and identify the controlling entity. Construct the entity interaction diagram to show the claiming sequence of the event upon the entity behaviors, and the coordination of messages passing between entities.

6: Refine interaction diagrams. Introduce the high level behaviors necessary to maintain the list of information stored by the system.

7: Queries and reports. Introduce the events necessary to enquire and report upon the state of the abstract system.

8: Errors. Consideration of the effects upon the system of reversing or changing any event. Consideration of events arriving in the wrong sequence. Missing events.

9: Physical implementation. By consideration of the volumes and frequencies estimated for each event, decide upon the physical environment in which the system is to be implemented. The practical coding of the system and the possible optimizations that can be effected in the chosen environment.

3.10.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

The developer stated that DSSAD is well-suited for time-critical or real-time applications, and data processing or database projects. Types of applications for which delivered systems have been developed with the method include classical order entry, stock control, invoicing system using on-line facilities, production of responses to magazine reader inquiries, and training management. The estimated number of such delivered systems is less than five, in the same number of organizations. The method is intended for projects of all sizes; it has been used for projects in the small to medium range. COBOL is the implementation language most frequently used when coding systems developed with the method.

Target Constraints

The method does not prescribe steps for handling such constraints.

Modes of Expression (Tables 9,10)

DSSAD requires specified documentation templates and strongly encourages narrative overviews of modules, structured English, and program design language. Required iconographical modes of expression include data-flow diagrams, entity-relaationship diagrams, and hierarchy charts. There are several modes of expression the developer considered inconsistent with the method; see the associated tables for specific modes.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

Incremental or evolutionary development and executable specifications are techniques utilized by DSSAD in order to clarify system requirements. Analysis and review techniques include data-structure analysis, design reviews, and code walk-throughs.

Other Technical Aspects

A concept considered essential to the method is structured programming. The developer stated that concepts such as abstract data-types and module coupling or cohesion are compatible with DSSAD.

3.10.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

The method provides a framework in which complexity may be assessed; other activities associated with project management are not addressed.

Communication Channels

Walk-throughs are conducted with the user at the end of Steps 1, 3, 5, 7 and during Step 9, described earlier. Continuous involvement of the user takes place throughout the process.

Quality Assurance (Tables 12,14,15)

The developer reported that error analysis rules and techniques incorporated into the method ensure the detection of all errors possible at each stage.

Documentation Formats (Table 16)

The method requires a series of documents to be produced. Those documents whose formats are prescribed by the method are tailorable within the method. For the specific documents, see the associated table.

3.10.5 EASE OF USE

Technology Insertion

In the developer's opinion, a development team leader, in order to use the method successfully, would need a bachelor's degree or two to three years of college-level technical education, no prior development experience, working knowledge of one programming language, and experience with one software system.

Training is available in the form of overview presentations, classroom tutorials, on-site consulting by the vendor, and user manuals. Five to ten days would be required for project managers to understand the major features and benefits of DSSAD, as well as for experienced developers to learn to use the method's essentials. Two months would be required to bring an experienced developer to the level of expert user.

Automated Facilities

Not applicable.

3.10.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

Not applicable.

Acquisition Costs

Technical training cost is $4,000. A management overview costs $1,000.

Contact Information

Colin Knight 0734 470440
23 Carlton Road
Caversham, Reading R447NT
England [Developer]

3.10.7 REFERENCES

[Knig81] C. Knight, L. Robertson, and L. Pink, "A Practical Implementation of Data Structured Systems Analysis and Design", Reader Enquiry Serivce, Jackson Method User Group, Amsterdam, Sept. 1981.

3.11 E-DEV/ESA -- Essential Systems Development/Essential Systems Analysis

3.11.1 BACKGROUND

Synopsis

Essential Systems Development (E-DEV) provides a framework for system development, based on the framework originally developed as part of Essential Systems Analysis (ESA), with strategies designed to integrate requirements definition productivity techniques and tools. ESA addresses the activities of requirements definition and specification, as well as preliminary design. E-DEV expands on ESA's treatment of design yielding the design technique Essential Systems Design (ESD), and adds a data modeling front-end, Quintessential Systems Analysis (QSA). The E-DEV managerial strategy expands on ESA's leveled systems development concept, integrating data modeling, prototyping, and Joint Application Development (JAD).

[Authors' note: E-DEV was developed as an extension to ESA by one of its co-developers, but ESA continues to be used without these extensions. They are discussed together here and listed separately in the tables in Chapter 6.]

History

Essential Systems Analysis is based upon DeMarco's structured analysis. Essential Systems Development incorporates techniques of information modeling, logical database design, Yourdon/Constantine's structured design, and IBM's JAD. With ESA [McMe84], Stephen McMenamin and John Palmer introduced a modeling approach based on separating the essence of a system from the details of its implementation and caused modifications of Yourdon's SA/SD, to deemphasize the practice of modeling the existing system. ESA was first used on a deliverable system in 1980, and E-DEV in 1981.

3.11.2 DESCRIPTION

Essential Systems Development has three sub-methods: Quintessential Systems Analysis, Essential Systems Analysis, and Essential Systems Design. QSA provides strategies for developing data models, ESA provides strategies for developing logical process models, and ESD provides strategies for integrating logical data and process requirements with implementation technology. E-DEV strategies are organized using both the waterfall and spiral development process paradigms.

Taking a waterfall view, the E-DEV technical strategy presents the logic of systems development and requirements definition. First the data model is developed using QSA. The output of QSA is transformed into a logical process model through the application of ESA. Both logical process and data requirements are then integrated with technology through the application of ESD.

Taking a spiral view, the E-DEV managerial strategy begins a project with the development of a first-cut data model, logical process model, and implementation model using parts of QSA, ESA, and ESD. The resulting models form the basis for the creation of a system prototype. They are also the basis for estimating the complexity of detailed requirements definition work and for precise establishment of project scope. Detailed requirements definition follows, during which the various models and the prototype are updated as additional requirements are identified. The final system is either a fully evolved prototype or one coded from the combined detailed passive requirements models and the prototype.

Essential Systems Analysis views the system being developed in terms of two different types of system models - the "Essential Model" and the "Incarnation Model". The creation of the Essential Model focuses on analysis of the major activities of the system from a logical view - what the system must do. The Incarnation Model is an extensive implementation environment requirements model. It provides a more physical view of the system and its environment - how the system will be built.

Continuing the process, ESD consists technically of two activities: specifying the implementation environment and mapping the essential requirements to the implementation environment model. The data flow diagrams and entity-relationship diagrams are redefined, along with state transition diagrams, to show the allocation of the essential model to processors. Based on this allocation, decisions are made related to the concurrency requirements of each processor, and the system support environment for the application is modeled.

Next the code organization phase incorporates the techniques of Structured Design(SD) to perform initial design and detailed design. An object-oriented design approach is also used to convert fragments of the essential model to program designs. As in SD, lower level data flow diagrams are created using successive refinement to produce hierarchical sets of diagrams at increasing levels of detail. Structure charts are used to define the physical structure of the program units of the system. They show the calling hierarchy, that is, what program units call or are called by what other units. Created by transform analysis from data flow diagrams, they are refined based upon the aspects of coupling, cohesion, complexity and reusability of individual models. The process is also divided into an external and an internal design stage. As part of internal design, database and program design are integrated.

3.11.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

ESA has been used in 51 to 100 organizations for developing 21 to 100 delivered systems, while ESD has been used in 5 to 20 organizations for developing 5 to 20 systems. Both have been used on projects of all sizes, including municipal bond insurance, utility customer service, pension fund account management, and investment/debt portfolio management. The languages most frequently used have been COBOL and C.

The method is best suited for data processing or database systems. The developers indicated that it is also well-suited for systems programming and distributed processing and it is compatible for use in embedded systems or process control, scientific or engineering applications, time critical or real-time applications, or expert systems.

P. Ward [Ward86] states that McMenamin and Palmer's extensions to SA/SD apply to transaction-oriented business systems. The extensions provided in the Ward/Mellor method include specific techniques for generating the Essential Model and the Implementation (Incarnation) Model for real-time applications.

Target Constraints

The method addresses details of the target configuration in the Incarnation Model and isolates portability concerns through the distinction between the Essential Model and 'he Incarnation Model. While ESA identifies issues concerning the target hardware architectures and the operating system, ESD addresses these and other target constraints in greater detail.

ESD provides support for the selection of target hardware architectures. All features are considered within a cost/benefits framework. ESD treats target operating systems in a manner similar to hardware architectures. In addition, opportunities for the expansion of system software are identified.

Entity state definitions in the "Q"/data model and response synchronization activities in the essential model are E-DEV's way of specifying technology independent timing constraints. ESD provides time-windows for I/O and mandates the specification of maximum response times for events.

ESD allows the implementation environment model to contain standard fault tolerance design templates, including transaction logging, parallel updates, backup systems, and mirror files. ESD's implementation environment model also contains standard access security design templates, such as password, security profile, and hangup/dialback.

E-DEV addresses portability through its division between internal and external views of the system.

Modes of Expression (Tables 9,10)

ESA primarily uses the data-flow diagrams of structured analysis and the entity-relationship diagrams of information modeling to express and refine the system requirements. It calls for the use of structured English to provide user-friendly narrative overviews of procedures, subroutines, and packages, in the form of mini-specs for the components shown in the structure charts. It encourages the use of hierarchy charts, finite state diagrams, and control-flow diagrams.

E-DEV's modes of expression extend beyond those of ESA. Its use of entity-relationship based event models, entity-state diagrams, and a data model context diagram and its requirements for hierarchy charts and system flowcharts are the major *differences.*

ESA and E-DEV provide mapping rules for translating from one mode of expression to another. ESA provides rules for translating from:

- physical data flow diagram (DFD) to logical DFD,
- entity/relationship diagram to DFD,
- entity/relationship diagram to data structure diagram,
- data flow diagrams to module hierarchy.

E-DEV adds rules for translating from:

- context diagram to entity/relationship diagram,
- entity/relationship diagram to entity state transition diagram,
- entity state transition diagram to DFD,
- entity state transition diagram to process description,
- attribute definition to DFD,
- attribute definition to process description,
- process description to module hierarchy,
- process data dictionary to module hierarchy.

Essential systems development views requirements definition as a series of requirements transformation activities. Therefore, the developer states that the whole of E-DEV is geared toward facilitating the transformation across phases of systems development in general and requirements definition in particular.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

Both ESA and E-DEV require rapid prototyping, data-structure analysis, data-flow analysis, control-flow analysis, decision tables, and JAD. They encourage incremental development and Change Control Board review. Although simulation is not addressed, an animated perspective of the static models could be used as an aid to conceptualization. ESA encourages entity-relationship analysis, while E-DEV requires it, as well as requiring design reviews and encouraging the use of formal proof techniques and code walk-throughs.

ESA uses two requirements perspectives, essential/logical and implementation/physical. E-DEV adds a third perspective, the Quintessential view ("Q"), and then divides requirements into three major views, data/quintessential, process/essential, and design/implementation.

E-DEV's use of event partitioning provides a common unit across all requirements transformations. A given event will identify a well-defined portion in each of the models. Therefore, transformation of requirements can be partitioned by event: one event's worth of essential model is derived from that event's worth of quintessential model; one event's worth of implementation model can be derived from that event's worth of essential model.

Other Technical Aspects

E-DEV uses the techniques of structured analysis, structured design, and entity-relationship analysis. The developer states that elements of these techniques minimize the cost of changes in requirements. E-DEV supports the early detection of inconsistencies and/or errors through its leveled systems development approach with the technique of "blitzing", a refinement of JAD. Each method within E-DEV, QSA, ESA, and ESD, has a blitzing strategy: a procedure for the rapid development of a first-cut model of a given type. These strategies are used early, i.e., in the feasibility study phase for a project.

The preliminary design resulting from the technique of blitzing is done by a group of developers and users in a workshop setting. The appropriate models are built with the passive modeling tools of information modeling, structured analysis, and structured design. Leveled system development also advocates the construction of active requirements models through a variety of prototype types. It is stated that these passive and active models make inconsistencies almost immediately clear.

In addition to addressing portability issues, the separation of concerns between the Essential Model and the Incarnation Model is intended to assist in reducing the effort required to adapt end-product systems to new applications.

3.11.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

The E-DEV managerial strategy is an expansion of ESA's leveled systems development concept. The developer indicated that its techniques for analysis of business and technological problems provide a basis for analyzing risk, assessing cost and complexity, tracking project progress, project planning, and scheduling and allocation of personnel and other development resources. The techniques of function point analysis [Albr84] have been used for assessing complexity on projects under development with this method.

E-DEV/ESA

Commun'cation Channels

E-DEV and ESA provide a medium for communication of the system requirements to management and users and within the development team. The visual representations of the existing system and/or the proposed system with data flow diagrams are in terms of processes and information that is meaningful to the user [Stev84]. Utilization of other visual representations, Structured English mini-specs, and a data dictionary provides coordinated communication within the development team and relevant sectors of the development organization.

Essential systems development uses blitz meetings, one-on-one interviews, and client walkthroughs to involve the client in the development process. During the feasibility study, E-DEV includes the software client in blitz meetings. Later in the project, one-on-one meetings are used to clarify specific issues with specific clients. Throughout the project, client walkthroughs of models and prototypes are held.

Client walkthroughs of models and prototypes are held to assure quality and conformance to actual needs. The list of identified events coordinates sessions of blitz meetings and walkthroughs. The work group in a blitz meeting establishes requirements for a given event. The review group in a walkthrough reviews the specified requirements for a given event.

The developer states that communication between team members is facilitated through the method's provision of a precise and comprehensive vocabulary, as well as detailed principles and heuristics from well-known and new techniques.

The event-orientation is thought to facilitate communications between management and the technical team. At each stage of requirements definition, requirements can be partitioned into event-based units. These units are the basis for requirements complexity estimation, team/team member productivity estimation, team/team member work assignments, and an overall project plan.

Quality Assurance (Tables 12,14,15)

The method requires testing activities and maintenance of records of technical decision-making. E-DEV supports the early detection of inconsistencies and/or errors through its leveled systems development approach, including blitzing, walkthroughs, and prototyping. Each method within E-DEV has a blitzing strategy, as described previously. Involving the user in a group-workshop approach for blitzing and walkthroughs facilitates the early discovery of inconsistencies that would not be found during one-on-one interviews. Additionally, the developer states that the passive models from information modeling, structured analysis, and structured design, combined with the active models from prototyping, make inconsistencies almost immediately clear. Although the "direct" technical strategy is preferred in E-DEV, the original strategy from ESA ("archaeological") is still utilized, at least in part, since it provides for essential model "reality testing" and quality assurance.

Documentation Formats (Table 16)

The textual and graphical information required by this method provides a basis for internal documentation of the system as it is proposed, evolves, and is finalized. Some of the system models, in particular the higher level diagrams, are suitable for insertion in deliverable documentation.

3.11.5 EASE OF USE

Technology Insertion

The minimum qualifications needed by a development team leader for use of the method include a bachelor's degree, three to five years of development experience, a working knowledge of one programming language, and development experience with at least one previous software system. For successful use, a lead developer should have an understanding of finite-state machines, entity-relationship modeling, parallel processing, object-oriented programming, functional decomposition, virtual machines, data independence, and device independence.

Training and information is available through the developers, through users' groups such as the Structured Methods Forum and those associated with various CASE tools, and through many independent consultants. Types of training include:

- hands-on demonstrations;
- overview presentations;
- classroom tutorials from Atlantic Systems Guild, Technology Transfer Institute, and others;
- user manuals;
- on-site consulting;
- related publications, including Atlantic Systems Guild newsletters.

The developers estimated that 3-5 days would be required for a project manager to acquire an understanding of the major features and benefits of the method, while it would take an experienced developer 10-30 days to learn the essentials and 6-12 months to achieve the level of expert user of the method.

Automated Facilities

The method does not require the use of an automated tool: however the techniques of the method, such as data flow diagrams, entity-relationship diagrams, the data dictionary, and mini-specs are supported by many CASE tools. These tools would identify many of the inconsistencies that can arise as a result of specification definition and changes.

The available CASE tools which support QSA, ESA, and ESD include:

Product	Vendor
Excellerator	Index Technologies
DesignAide	NASTEC
PROMOD	PROMOD
Software through Pictures	Interactive Development Environments
Teamwork/IM,SA,SD	Cadre Technologies
S. E. Workbench	Yourdon
Information Engineering Workbench	KnowledgeWare
Information Engineering Facility	Texas Instruments

3.11.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

The hardware/software configuration depends on the CASE tool selected.

Acquisition Costs

The cost per person for technical training is estimated to range from $2,000 to 10,000 and for a management overview from $2,000 to 3,000. The costs for software tools are available from the individual tool vendors.

Contact Information

John F. Palmer — 914-472-9337
The Atlantic Systems Guild, Inc.
55 Walbrooke Road — [Co-developer of ESA,
Scarsdale, NY 10583 — developer of E-DEV]

Stephen M. McMenamin
c/o The Atlantic Systems Guild, Inc. — 212-620-4282
353 W. 12 Street
New York, NY 10014 — [Co-developer of ESA]

Technology Transfer Institute, Inc.
741 Tenth Street
Santa Monica, CA 90402-2899 — [Provider of seminar]

3.11.7 REFERENCES

[McMe84] S. M. McMenamin and J. F. Palmer, Essential Systems Analysis. New York: Yourdon Press, 1984.

[Palm87] J. F. Palmer, "Integrating the Structured Techniques with JAD: Leveled Systems Development", presented at the Structured Methods Conference XII, Chicago, Aug. 1987.

[Palm88] J. F. Palmer, "CASE and ESA", Proceedings of the 14th Semiannual Seminar of the NY Chapter of the IEEE/CS, November 1988.

[Palm89] J. F. Palmer, Essential systems Development: A Fourth Generation Methodology, Seminar available at Technology Transfer Institute, Inc., Feb. 1989.

[Albr83] A. J. Albrecht and J. Gaffney, Jr., "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation", IEEE Transactions on Software Engineering, Volume 9, No. 6, November 1983, pp. 639-648.

Further information about methods which have been incorporated into E-DEV and ESA can be found in [DeMa78], [Stev74], [Ward86], [Your79], [Your86a], [Your86b].

3.12 GYPSY -- Gypsy Methodology

3.12.1 BACKGROUND

Synopsis

The emphasis of this method is on the construction of highly-reliable software systems. The method consists of techniques for specifying, programming, and verifying in an environment supporting these techniques so that they may be applied in practice, and uses the GYPSY language for programming and formal specification. Predicate calculus is used for verification. The method is available for general use but the U.S. Government must approve foreign export.

History

This method was first used for the development of a deliverable system in 1979; its principal architect was Donald I. Good. It is based on methods of program verification, and has been described as one of the more mature mechnancal program verification system.

3.12.2 DESCRIPTION

GYPSY, the Gypsy Methodology, prescribes specific procedures for accomplishing system specification, system design, system implementation, software quality assurance, and formal verification/proof of correctness. The concepts of process abstraction and use of assertions are essential to the method.

Gypsy integrates three approaches to formal specification. These are 1) program assertions, 2) state machines, and 3) algebraic axioms. Any of these or a combination may be used. The Gypsy language itself incorporates both specifications and programs, and may be called a program description language.

The method results in a system and its formal specification in the form of a number of Gypsy components called units. Program units are intended to be specified and verified independently by the method. Large programs are built from these smaller units.

Verification methods used by Gypsy include deductive proof, run-time validation, and conventional testing. The deductive proof method consists of a set of well-defined rules for reducing a program and its specifications to a set of theorems, or verification conditions. These verification conditions are sufficient to demonstrate that all program executions meet their specifications. The run-time validation method evaluates the specification as the program runs, to determine whether a precisely defined set of program states is true. If the specification is found to be false in any of the states of the set, then an exception is raised.

The Gypsy Verification Environment (GVE) consists of a number of tools to support the development of large verified systems and maintains a Gypsy library of such components as programs, specifications and verification conditions.

3.12.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

The respondent regarded the method to be well-suited for scientific or engineering applications, systems programming, distributed processing or networks, and data processing or database. Originally developed for communications processing applications, Gypsy is primarily used for development of secure systems. It is one of two methods endorsed by the National Computer Security Center for this purpose. It has been used, for example, in Honeywell LOCK, Honeywell SCOMP, and TRW's AMPE, as well as on a number of other security-related projects. The method is intended for use on projects of medium size; it has been used both on small and medium projects. The estimated number of delivered systems developed with Gypsy was between five and twenty, within the same number of organizations. Gypsy and C are the implementation languages most frequently used when coding systems developed with the method.

Target Constraints

The explicit variable and data structure allocation allows for handling of spatial constraints of the target system. Formal modelling of hardware addressed special features of the target hardware architecture. The method has explicit features for expressing concurrency, and it is possible to model secure systems using the method.

Modes of Expression (Tables 9,10)

Two textual modes of representation are required by the method and each is provided with automated support. They are a program design and formal specification language. The specification language is designed to allow successive refinement of specification and code, with proofs possible at each level of abstraction.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

Formal proof techniques are essential: rapid prototyping and incremental development are strongly encouraged to clarify system requirements or behavior.

Other Technical Aspects

The method is seen as capable of reducing the effort needed to incorporate changes in the requirements by enforcing modularity, thus localizing the effect of changes. The formal verification of conformity between code and specification assists in ensuring consistency between these entities when changes are made. Each module is self-contained and relevant features are fully defined in the external specification/interface specification, which is seen as helping with the identificaiton of potentially reusable components.

3.12.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

The method supports specification and proof at arbitrary levels of abstraction. Project planning may be developed around a very high level system specification. Otherwise, Gypsy does not address project management activities.

Communication Channels

Gypsy enforces very exact and very clean interface specifications. The interfaces are sufficient to fully characterize all relevent aspects of a module. This feature is designed to facilitate communication within the development team, and may be relevant between management and the technical team as well. High-level specifications are said to be quite readable, and thus can facilitate communication between the client and the development organization.

Quality Assurance (Tables 12,14,15)

The method provides automated support for formal verification techniques. This is seen as reducing the burden of testing by catching design and logic errors early in the software development cycle.

Documentation Formats (Table 16)

The system specification is in the form of Gypsy code, i.e., Gypsy is a specification and programming language combined.

3.12.5 EASE OF USE

Technology Insertion

The minimum qualifications needed by a development team leader for successful use of the method were given as a bachelor's degree, one to two years of development experience, knowledge of two programming languages, and experience with two different software systems. Theoretical constructs which should be understood by an experienced developer are formal logic and proofs. The respondent commented that facility with mathematics, programming, proof techniques, and familiarity with Pascal-like languages such as Ada would be helpful as well.

Assistance to train an organization in the use of Gypsy is available through hands-on demonstrations, overview presentations, classroom tutorials, on-site consulting by the vendor, an on-line help facility, user manuals, and a users' support group. It was estimated that one day would be required for a project manager to acquire an understanding of the major features of the method, and five days for an experienced developer to learn to use the essentials. Three to six months would be required for such a developer to achieve expert user level.

Automated Facilities

The method is supported by the Gypsy Verification Environment (GVE) from Computational Logic, Inc. Included are tools such as a Gypsy parser for checking syntax and semantics, a Gypsy verification condition generator, an interactive theorem prover, an automatic algebraic simplification system for the predefined data types of Gypsy, a Gypsy to Bliss translator, and a Gypsy to Ada translator. Additionally, there are tools supporting incremental development of programs and their verification, a data dependency tool for analyzing the specifications of a trusted computing base, and information display, help, and session logging facilities.

3.12.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

A Sun 3-60 host with 12 mb of memory and 45 mb of swapping space is appropriate for the tools of the method.

Acquisition Costs

Unit costs to acquire the method and required components are $12,000 for a single user but reduced to $6,000 for multiple users. Technical training unit costs are $5,000 for a single user and $2,000 for multiple users; with these courses, there is a management overview included.

Contact Information

Computational Logic, Inc. 1717 West Sixth, Suite 290 Austin, TX 78703	512-322-9951 [Provider]

3.12.7 REFERENCES

[Good85] D. I. Good, "Mechanical Proofs about Computer Programs", in C. A. R. Hoare and J. C. Shepardson, eds., Mathematical Logic and Programming Languages. Englewood Cliffs, NJ: Prentice-Hall, 1985, pp. 55-75.

[Good78] D. I. Good, R. M. Cohen, C. G. Hoch, L. Hunter, and D. F. Hare, "1978 Report on the Language Gypsy, Version 2.0", Technical Report ICSCA-CMP-10, Certifiable Minicomputer, Project ICSCA, The University of Texas at Austin.

3.13 **HOOD** -- Hierarchical Object Oriented Design

3.13.1 **BACKGROUND**

Synopsis

HOOD is an architectural design method oriented towards the development of Ada programs. The identification of target architecture is supported, leading to detailed design where objects, in the object-oriented sense, are further designed by means of an Ada based program design language. The method is applied primarily to the preliminary and detailed design phases of the software process.

History

The method is a merging of the Abstract Machine concept used by MATRA ESPACE and the concepts of Object-Oriented Design. Hood was developed by the European Space Agency Technical Directorate as an Architectural Design Method for software to be programmed in Ada. It was first used with respect to a deliverable system in 1987, and has since been selected for the Columbus Manned Space Station Program and the HERMES Manned Spaceplane program, which are independent projects. The European Fighter Aircraft project has also mandated HOOD and Ada for its contractors.

3.13.2 **DESCRIPTION**

Hierarchical Object Oriented Design (HOOD) has resulted from merging methods known as Abstract Machine and Object-Oriented Design. The concept of machine in the first is similar to that of object in the second. The abstract machine enforces a hierarchical structure where the other method did not. On the other hand, the developers believe that the object-oriented method enforces the design of more coherent objects.

The developers addressed a need for a way of distributing the development of large systems among several organizations by establishing two hierarchies:

1. The seniority hierarchy, present in the abstract machine method, permits senior objects to control and use junior objects. Layers of objects are thus established with high cohesion and low coupling.

2. The parent-child hierarchy allows an object to be the composition of other objects. This is basic to the concept of subcontracting software objects to different organizations.

In the developer's opinion, productivity is improved by means of a consistent, standard top-down method of decomposition. The problem domain is successively mapped into the design and implementation, by modelling the real world entities as a set of OBJECTS. These objects may in turn be decomposed further, into objects that become more software oriented, called Abstract Data Types. HOOD supports the software engineering principles of Abstraction and Data Hiding, so that Ada packages are built that encapsulate the data (state of the objects) and provide access only through operations or procedures/functions. This, the developer feels, leads to an inherently more maintainable system, with considerable benefits in the software integration stage. The developer mentions that this result has been borne out on many Ada development projects. The developer also feels that the consistency checks that are built into the decomposition and interface definition process result in quality benefits. HOOD emphasizes the early and clear definition of interfaces, and these are checked by the HOOD Toolset.

HOOD can be used for prototyping at the design stage by means of an environment package and early definition of the major objects.

3.13.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

The developer rated the method well-suited for use with embedded systems or process control, time-critical or real-time systems, scientific or engineering applications, and distributed processing or networks. Examples of applications for which delivered systems are being developed using the method are space system software, a general check-out system for ground support software, and real-time systems including data acquisition and processing, avionics systems, telemetry, and control-command systems. The method is intended for use on medium and large-sized projects; it has been used on projects of all sizes, with an estimate of between five and twenty systems developed using HOOD within the same number of organizations. Implementation languages most frequently used are Ada, which is the first target, and C or FORTRAN which are secondary targets.

Target Constraints

The method prescribes steps for handling timing constraints, spatial constraints, and special features of the target hardware architecture and operating system. This is accomplished by describing non-functional constraints as informal comments in a field of the document template for each object. Concurrency and fault-tolerance issues are treated in the same way.

Modes of Expression (Tables 9,10)

Textual modes of representation include narrative overviews of modules and a program design language, both of which are required. Specified documentation templates are strongly encouraged. All three of these textual modes are provided with automated support.

Required iconographical modes are data-flow diagrams and control-flow diagrams. Strongly encouraged are finite-state diagrams, petri nets, and ESTEREL Formalism.

The method facilitates the transformation across phases of the software process by using informal text description in a natural language before going to formal descriptions, PDL, and code. In the developer's opinion, the mixing of informal text with formalized notation will allow easy introduction of formal notations, so that HOOD will be adapted as a back-bone framework in the use of advanced techniques.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

The method requires the use of incremental or evolutionary development to clarify system requirements. Techniques required are design reviews, code walk-throughs, and Change Control Board reviews.

Other Technical Aspects

The use of information hiding helps to localize changes, which is seen as assisting with reducing the effort needed to fully incorporate changes in the requirements. The use of reverse engineering techniques, the use of a data dictionary, configuration control and management all help to address the concept of traceability between requirements and design, and between code and design.

The use of tools associated with HOOD allow import from or export to the tools components such as design hierarchies or parts of them, thus potentially allowing reuse of other designs or design parts.

3.13.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

General guidelines are provided for allocation of personnel to tasks, and allocation of development resources. Configuration management is required but not directly addressed. Other project management activities are outside the scope of the method.

Communication Channels

Within the technical development team, the use of graphical and textual formalisms applied to object entities through the description templates is seen as facilitating communication, as well as the informal natural language descriptions of objects. Between management and the development team, communication is facilitated by the method's definition of phases in the development, documents delineating milestones and checkpoints, and the QA procedures for documents.

Between the software client and the development organization, the client is involved in the software process and in QA by being involved in delivery of informal text documents, design prototyping (a milestone), and reviews of documents.

Quality Assurance (Tables 12,14,15)

The method separates in time the step where an object interface is identified (specified) externally as a child connected to its siblings, from the step where the object is implemented. QA procedures and verification techniques are based on this key feature, leading, in the developer's opinion, to the improved verification of interfaces. The method provides automated support for prescriptive checking of interfaces, and provides a framework for generation of tests based on system requirements and unit/integration testing.

Documentation Formats (Table 16)

The method requires a number of documents whose formats vary in tailorability. For more detailed information, see the associated table.

3.13.5 EASE OF USE

Technology Insertion

Minimum qualifications needed by a development team leader for successful use of HOOD were given as two to three years of college-level technical education, one to two years of development experience, knowledge of two programming languages, including knowledge of Ada, and experience with three to four different software systems. In addition, major theoretical constructs which should be understood by an experienced developer are data-flow diagrams, as well as state transition diagrams and related extensions or tables.

Training is available with hands-on demonstrations, overview presentations, on-site consulting by the vendor or independent consultants, user manuals, and a users' support group. The developer estimated that a project manager would need three days to acquire an understanding of the major features and benefits of the method, while an experienced developer would need 10 days to learn to use the method's essentials. Depending on the complexity of the projects worked on, an experienced developer would need three to six months to achieve expert user level.

Automated Facilities

The developer listed several tools provided by external vendors which support activities of the method. STOOD by TNI supports graphical, textual, and syntactic editing, Ada generation, configuration management, and design checking. The Hood Toolset by Software Sciences LTD, VSF by Systematica, and Adanice by Intecs all support syntactic edition, configuration management, and design checking.

3.13.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

Appropriate configurations are Sun/UNIX workstations, VAXStations/VMS, Apollo/Aegis, and partial support on PC compatibles.

Acquisition Costs

The method is in the public domain. Cost for technical training is $15,000 of a single user, $10,000 for low-volume users, and $8,000 for high-volume users. Unit cost for a management overview is $20,000 for single and low-volume users; for high-volume users, the cost is $10,000.

Contact Information

Cisi-Ingenierie
2, rue Jules Vedrine
31400 Toulouse
France

(33) 6120.4324

[Provider]

3.13.7 REFERENCES

[Heit88] M. Heitz, "Using Hierarchical Object Oriented Design for the Development of Distributed Ada Systems", Proceedings of Munich Ada-Europe 1988 Conference "Ada in Industry", Cambridge University Press, Ada Companion Series.

The Hood manuals are being revised and updates will be forthcoming in April and July 1989 from the European Space Agency.

3.14 **IBM/4LDM** -- The Four Level Design Method

3.14.1 **BACKGROUND**

Synopsis

IBM/4LDM provides a method for achieving a fully-targeted detailed design from the specifications for a software system. It uses an Ada-based design language to produce and record the design at various levels of abstraction. Use of Ada as a design language allows the use of an Ada compiler and related tools to support semantic and syntactic checking of the design. The method is available for general use.

History

IBM/4LDM was developed by Don O'Neill, previously of the IBM Federal System Division. It is based upon software engineering practices developed at that organization in the late 1970's. The method was invented and first used in 1979.

3.14.2 **DESCRIPTION**

IBM/4LDM, the Four Level Design Method, incorporates the use of an Ada-based design language into the Software Factory approach used at IBM Federal System Division (FSD). The developer states that the use of Ada as a design language permits "the design to obtain rigor in syntax and semantics through the use of the Ada compiler product tools," and that its use "provides a platform for systematically accomplishing rapid prototyping through use of the emerging software design and product itself." [ONei86]

IBM/4LDM is founded on a functional decomposition approach and is well-suited for use within the context of several software process paradigms; however, it is not dependent on one particular paradigm. Top down or bottom up design can be applied. Essential to the method are the concepts of stepwise refinement, information hiding, process abstraction, abstract data- types, and structured programming. The respondent reported that, as a result of using the method, testing is reduced. It was felt that errors are dramatically reduced when systematic design is practiced.

The Four Level Design Method was adapted from software engineering practices used at FSD. These practices have a goal of producing and verifying modular designs and structured programs. The design stages ensure correspondence of design to specifications, provide for functional allocation and decomposition of procedures and data, and elaborates the design through the use of stepwise refinement, a program design language, and correctness techniques.

The four levels of IBM/4LDM are:

- Level 1. Create the user contract, where the user is considered in the same terms as other software products which might utilize or interface with the system being designed;
- Level 2. Portray design parts and their relationships, both data interfacing and tasking;
- Level 3. Elaborate a detailed functional design which is independent of the target architecture or operating system;
- Level 4. Provide detailed designs which are fully targeted to the operating system and instruction set, and ready for consideration of efficiency and capacity constraints during implementation.

Specific templates can be used to record each level of design. Furthermore, each level should be communicated in such a way as to ensure understandability with the intended audience, whether that be technical or non-technical personnel.

For purposes of management, levels 1 and 2 correspond to the specification review milestone, the Preliminary Design Review. Levels 3 and 4 correspond to the design review milestone, the Critical Design Review. Relative to MIL-STD 2167, level 1 and 2 designs are included in the Software Top Level Design Document, and level 3 and 4 designs are included in the Software Detailed Design Document.

In level 1, the specification is input to the process. The Ada package specification is used to express the design. The design is reviewed by trained experts who must ensure that the design meets the criteria for completeness, correctness, usability, performance, and overall user satisfaction.

At level 2, the design for each of the level 1 components is recorded as an Ada package specification and package body. Tasking constructs and abstract data types are used to express the design. Procedural elaborations are not carried out in this level, but simply appear as procedure calls. These level 2 Ada packages are evaluated for possible reuse.

In level 3, procedure elaboration is used to express detailed functional design. In using the Ada design language, the outer syntax function expressions, and the inner syntax data refinements must be specified. In order to control the quantity of Ada design language being produced, level 3 may be limited to only those procedures whose calls were specified in level 2.

At the final level, procedures and functions are sufficiently elaborated to provide a fully targeted, detailed design document. At level 4, the product should conform fully to MIL-STD 1815A.

3.14.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

The method is reported to be well-suited for several application areas, including embedded systems or process control, time critical systems, scientific or engineering applications, distributed processing, and data processing.

An estimated 5 to 20 delivered systems have been developed using this method, with the same number of organizations having used the method. Both medium and large-size projects have been developed; however, the method is intended for use on large projects.

The Four Level Design Method is based on Ada, and Ada has been used successfully as the implementation language. Jovial was used with the method unsuccessfully.

Target Constraints

At level 2 the method includes tasking and permits exceptions, which address concurrency and fault-tolerance issues. Other requirements of the target system are considered at level 4, which is target dependent. At this level the method addresses timing and spatial constraints, as well as special features of the target hardware architecture and operating system. By requiring target independence through Level 3, the method is intended to assist in portability problems.

Modes of Expression (Tables 9,10)

Required modes of expression include specified documentation templates, a program design language, and finite-state diagrams. Formal specification languages and mathematical notation are strongly encouraged. In preparation for implementation, the use of Ada in both high- and low-level design is staged.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

Rapid prototyping, incremental or evolutionary development, and executable specifications are encouraged as techniques to clarify system behavior. A number of analysis techniques are also encouraged, while design reviews and code walk-throughs are required.

Other Technical Aspects

Consistency of specification, design, and code as well as early detection of inconsistencies and/or errors are addressed by using Ada compilation of design.

3.14.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

Analyzing risk, assessing complexity and estimating initial cost are among the project management activities for which the method provides specific procedures. Other activities are addressed at a more general level.

Communication Channels

The four levels of IBM/4LDM correspond to different audiences. Level 1 provides the user interface specification for communication with the software client. Level 1 and also Level 2 are prepared for systems engineers and software engineers to read and understand. These two levels form the link between program managers and software engineers, through systems engineers. Level 3 designs are more detailed and intended for algorithm and data designers and software engineers. Level 4 designs are intended for implementing programs.

Quality Assurance (Tables 12,14,15)

Review techniques required by the method include design reviews and code walkthroughs. Facilitated Application Specification Techniques and Change Control Board review are encouraged.

Formal reviews are used in which the unanimous consensus of the reviewers is required. Use of the Ada compiler to check the design assists in identifying syntactic and semantic errors. In addition, the method incorporates an executable model, and uses informal proof of correctness techniques.

The method provides a framework for test planning at particular points in the software development process and for prescriptive checking of interfaces.

Documentation Formats (Table 16)

The following documents which are required to be produced have a format which is tailorable within the method: functional specification, architectural specification, interface specification, system structure chart, data dictionary, and design document.

3.14.5 EASE OF USE

Technology Insertion

A development team leader would need the following minimum qualifications to use the method successfully: a Bachelor's degree, 3 to 5 years' development experience, knowledge of two programming languages, and experience with 2 different software systems. Theoretical constructs which should be understood are finite state machines, and structured programming (prime programs, proper programs, data dictionary, disciplined data structures).

Classroom tutorials are provided for training purposes. A project manager could acquire an understanding of the major features of the method in one-half day. Experienced developers would need 2 days to learn the method's essentials, and 6 months to achieve expert user level.

Automated Facilities

Not applicable.

3.14.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

Ada-based development systems are appropriate for use with the method.

Acquisition Costs

There is no cost for acquiring the method, and no licensing policy.

Contact Information

Note: The acronym IBM/4LDM is not a standard product name.

IBM System Integration Division 301-493-1463
c/o John Rymer, Senior Programmer
6600 Rockledge Drive
Bethesda, MD 20817 [provider]

3.14.7 REFERENCES

[ONei86] D. O'Neill, "Software Engineering and Ada in Design", presented at the Washington Ada Symposium, March, 1986.

[ONei83] D. O'Neill, "An Integration Engineering Perspective", The Journal of Systems and Software, Vol. 3, 1983, pp. 77-83.

[ONei80] D. O'Neill, "Management of Software Engineering", IBM Systems Journal, Vol. 19, No. 4, Dec. 1980.

3.15 IEM -- Information Engineering Methodology

3.15.1 BACKGROUND

Synopsis

IEM is directed toward the development of information systems. Its goals include a reduction of excessive development time, effective support of business activities, and facilitating the incorporation of changes into a system. The method specifically addresses the activities of requirements definition, system specification, system design, software quality assurance, and project management, and stresses the involvement of the client. Use of fifth-generation, CASE, database and code generation technology is encouraged. The method is available for general use.

History

IEM was developed by Ian Palmer, James Martin and their colleagues. The basic techniques, which were termed "Data Analysis", were developed in Britain from 1973 through 1978. Another precursor of the method is D2S2, or Development of Data Sharing Systems. Further refinements involved providing a balance between data and activity analysis, clear mapping from business modelling (analysis) to design, adding a project management framework, and the creation of support tools. The current method was first used for a deliverable system in 1982.

3.15.2 DESCRIPTION

Information Engineering Methodology, IEM, is composed of seven stages. Each stage consists of tasks, techniques, and deliverables. Stage interconnections allow for alternate development paths, depending on the complexity of the project. At the center of the method is the Encyclopedia, a database which records all analysis and design decisions, progress, changes, and documentation. Varying views of this database are available to support communication, documentation, and control among developers and project managers.

The underlying concepts upon which IEM is founded include entity-relationship modeling, functional decomposition, development coordination of multiple projects within the same architectures, and separation of analysis from design. The method is well-suited to several software process paradigms and is compatible with the timebox paradigm. However, the most effective use of the method is dependent upon a paradigm which emphasizes a balance between data and process analysis/design. The method itself emphasizes techniques which describe how analysis and design are done, rather than only what is produced. The developer states that, due to up-front emphasis on business modelling and user involvement as well as code generation, the activities of flow charting, structured programming, code walkthroughs and testing can be skipped.

The seven stages of IEM are information strategy planning, business area analysis, business system design, technical design, construction, transition, and production. The stages proceed in a top-down manner as data-oriented, stepwise refinements. The first four stages are focused on diagrammatic techniques beginning essentially with:

- Entity analysis -- identification of relevant objects and their interrelationships, ultimately leading to data structure design and data storage distribution design,

- Functional analysis -- identification of relevant activities and their dependencies, ultimately leading to a decomposition into primitives, the design of input/output layouts and human interaction, and the generation of code.

The final three stages incorporate the traditional activities of code generation, testing, performance analysis, hardware installation, and user training. All of these activities are directed by, and interact with, the results of the analysis and design stages.

The information strategy planning stage requires extensive collaboration between the client and the information staff. The major deliverables of this stage are:

- An information architecture, represented by an entity relationship model and a function dependency model;
- A system architecture, summarized through data-flow diagrams;
- A technical architecture, represented through hardware, software, and communication diagrams.

To assist project management, the following is also produced during this first stage:

- A problem analysis which identifies implementation difficulties and procedural limitations;
- A RAEW analysis which recommends division of responsibility, authority, expertise, and labor;
- Decomposition of objectives or a critical success factor analysis.

The purpose of the business area analysis is to refine the information aichitecture. Entity types, relationships and attributes are documented, normalized, and life cycles for the major entities are estimated. Functions are decomposed in further increasing detail. Selected end users are asked to review the business area models to ensure accuracy.

The business system design stage stresses client involvement. Interfaces are defined, including screen layouts, reports, and forms. Prototyping is used to achieve a better understanding of the system. In the next stage, the technical design is evolved from the technical architecture. In this process the hardware elements, the data storage structures, and the required software tools are defined. The software design is normally achieved through use of database management systems, and through either structured languages, non- procedural languages, or a code generator.

During the construction stage, modules are coded and the database structure is populated. It is expected that automated code generators are used during this stage. Verification is accomplished through a three-phase testing procedure. The transition phase includes deployment of the system at the client's site, training, and trial runs. The production stage involves performance monitoring, software tuning, and evaluation of actual costs/benefits in light of design objectives.

Automated tools are not integrated into the framework of the method, but use of CASE analysis and design tools, a database management system and automatic code generators is an integral part of the development process.

3.15.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

The developer rated the method well-suited for distributed processing or networks and data processing or database applications. Examples of specific project areas in which delivered systems have been produced using IEM are: financial control, insurance, inventory control, geological survey, marketing, banking, CASE software, laboratory analysis, and manufacturing. More than 250 delivered systems were estimated as having been developed with the method, used by an estimate of over 100 organizations. The method is intended for use on medium and large-size projects; it has been used on projects of all sizes. COBOL and 4GL's are most frequently used for coding systems developed with this method.

Target Constraints

IEM prescribes descriptions of special features of the target hardware architecture required, and guidelines for handling security of access. The method addresses concurrency issues by means of dependency diagrams and data structure paths. By making the application independent of configuration, the developer reports that the method assists in portability.

Modes of Expression (Tables 9,10)

The method requires a textual mode of representation called action diagrams. Required iconographical modes include data-flow diagrams, entity- relationship diagrams, and hierarchy charts. Also required are process dependency diagrams, dialog flow diagrams, and data structure diagrams, with entity decomposition diagrams and data analysis diagrams encouraged.

Among the above modes of representation are mapping rules within the method for translating from one mode to another. One mapping is from the subject area diagram to the entity-relationship diagram to the process logic and then to the action diagrams. Another mapping is from the function dependency to the process dependency to the data-flow diagram to the dialog flow. Other mappings exist as well. By means of code generation from data structure diagrams and action diagrams, as well as by matrix analysis, walkthroughs, and prototyping, the method is seen to facilitate the transformation across phases of the software process.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

A number of techniques are encouraged for clarifying system requirements, among them JAD (Joint Application Development). Required review techniques include data-structure, data-flow, and control-flow analyses, design reviews, Facilitated Application Specification Techniques (FAST), process logic analysis, and development coordination.

Other Technical Aspects

The developer considers that the use of the method's Encyclopedia, well- defined scoping, development coordination, and maintenance of the business models all contribute towards reducing efforts to incorporate changes in the requirements. Moreover, use of the Encyclopedia, consistency tests, and mapping from analysis to design are intended to assist in ensuring consistency among specification, design, or code when changes are made to these entities.

Reusable components are identified with the method by means of current systems analysis, the fact that the decomposition network is top-down, automatic comparison of information views, and re-engineering techniques.

3.15.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

Specific procedures for project planning and Development Coordination are incorporated in the method, which provides guidelines for addressing other management activities.

Communication Channels

Among members of the development team, communication is facilitated via the Encyclopedia, quality assurance activities, and diagram walkthroughs. Between management and the technical development team, JAD, prototyping, and milestone reviews are used. JAD and prototyping are also used to facilitate communication between the develoi ment organization and the client, as well as intensive executive planning sessions. In addition, the client is involved during the software development process with intensive analysis sessions and acceptance testing.

Quality Assurance (Tables 12,14,15)

Specific directions are provided by the method for test planning, unit/integration testing, and prescriptive checking of interfaces. Early detection of inconsistencies and/or errors is assisted by the method's use of matrix analysis, walkthroughs, and prototyping. The method does not specifically require that a record of technical decision-making be maintained.

Documentation Formats (Table 16)

The method requires that a number of documents be produced, with none of the formats fixed; the format is either tailorable within the method or is not prescribed. Tailorable formats for two types of documents additional to those shown in Table 16 are provided. These two documents are a Business Model and the RAEW matrix.

3.15.5 EASE OF USE

Technology Insertion

The developer estimated that a development team leader would need two to three years of college-level technical education, three to five years of development experience, and experience on one software system in order to use the method successfully. A project manager could acquire an understanding of the major features and benefits of the method in three days, while an experienced developer would need 10 days to learn to use the essentials of the method and 12 months to achieve expert user level.

A number of training procedures are provided by the developer and are listed in Table 18.

Automated Facilities

The developer mentioned three tools which specifically support activities of the method: IEW by Knowledgeware, IEF by Texas Instruments, and POS by CSA. The method also provides internal program documentation, generated automatically from data produced in other steps of the method.

3.15.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

An IBM PC/AT or an IBM 3900 or 3400 would be appropriate for hosting automated tools supporting the method.

Acquisition Costs

The licensing policy is a corporate license.

Contact Information

James Martin Associates Inc. 703-620-9504
1850 Centennial Park Drive
Reston, Virginia 22180 [Provider]

Note: IEM and Information Engineering Methodology are trademarks of James Martin Associates.

3.15.7 REFERENCES

A three-volume set from Savant Institute, Information Engineering, by James Martin, is now in publication by Prentice Hall.

3.16 ISAC -- Informations Systems Work and Analysis of Changes

3.16.1 BACKGROUND

Synopsis

This commercial method contains five development components which are named: Change Analysis, Activity Studies, Information Analysis, Data Systems Design, and Equipment Adaptation. The following activities are addressed by the method: problem definition, requirements definition, cost estimation, specification, project management, scheduling, preliminary design, prototyping, detailed design, configuration management, documentation, and field testing.

History

The ISAC method was developed at Stockholm University; A. Nilsson and M. Lundeberg are principal contributors. It is a method that primarly deals with the description and decomposition of business functions and belongs to the same tradition as Yourdon and Gane & Sarson. The method was first used for deliverable systems in 1968 and it was included in the methods surveyed in Methodman I [Free82]. Today, the method has its primary market in Sweden, Norway and the Netherlands.

3.16.2 DESCRIPTION

The Informations Systems Work and Analysis of Changes (ISAC) approach is that of functional hierarchy/decomposition. In addition, it is founded upon data flow and data structure-oriented approaches. It is said to be somewhat more formal than other methods that are based upon a similar approach. It is well-suited to the waterfall, incremental, and transformational models of the software process, as well as to the 4GL paradigm. However, its most effective use is not dependent upon any particular software process paradigm. Essential concepts to ISAC are stepwise refinement, process abstraction, structured programming, and module coupling/cohesion.

The method is not coupled to any specific target environment and is often used in combination with other methods, for example, entity-relationship modeling. The work products of the method listed below describe aspects of the software process addressed by ISAC. These include:

- Activity model of current situation;
- Activity model of chosen change alternative;
- Change plan;
- Activity model with information subsystems;
- Priority plan for information subsystems;
- Detailed information analysis model;
- Equipment-independent data systems model;
- Equipment-adapted data systems model.

Specific representations named by the method are called: Activity graphs, Information-flow graphs, Component graphs, and Data system design graphs [Lund81].

3.16.3 TECHNICAL ASPECTS

Applicability and Usa͜ᵹe (Tables 4,5,6)

The developer reported the method to be well-suited for applications in the areas of embedded systems or process control, time critical or real-time processing, scientific or engineering systems, and data processing or database systems. However, examples of delivered systems developed with the method are accounting, payroll, order entry, inventory management, production control, and banking/insurance systems. The method has widespread use: more than 100 organizations have used ISAC on more than 250 delivered systems. Although the method is intended for medium and large applications, it has been used for projects of all sizes. COBOL and 4GL-languages have been most frequently used for the implementation of systems developed with ISAC.

Target Constraints

The developer stated that the method prescribes steps for handling timing and spatial constraints as well as special features of the target hardware architecture; specific information was not provided on the nature of these steps. The method is seen to assist in porting end-product systems to different target configurations in the development area called "equipment adaptation".

Modes of Expression (Tables 9,10)

The method requires specified documentation templates and provides automated support for this mode. It also requires decision tables. Required iconographical modes include data-flow diagrams and hierarchy charts.

There are mapping rules within the method to assist in transforming from one mode of expression to another. These rules apply to the transformation from ISAC-graphs to JSP-diagrams. Transformation rules and illustrations via examples are provided between the phases of the software process.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

ISAC requires incremental or evolutionary development and executable specifications in order to clarify system requirements or behavior. It requires several analysis and review techniques, including data-structure, data-flow, and control-flow analyses, decision tables, and Facilitated Application Specification Techniques.

Other Technical Aspects

The method utilizes the computer tool "Graphdoc" to facilitate incorporation of changes in the requirements. During information analysis the step that addresses consistency between specification, design and code as well as early detection of inconsistencies and errors is called "analysis of completeness and consistency". The method also requires that a data dictionary and design document be produced.

Reusability is addressed in the development area called "data systems design".

3.16.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

Specific directions and procedures are provided within the method for estimating initial cost, projecting cost of completion, and providing incremental data about expenditures, as well as for scheduling and/or manpower loading and configuration management.

Communication Channels

The developer states that communication is facilitated among all concerned in the software development project by simple graphical documentation of the development work. ISAC involves the client by encouraging users to take an active part during the development activities of change analysis, activity studies and information analysis.

Quality Assurance (Tables 12,14,15)

The method provides guidelines for several testing activities. Automated recording procedures are provided for maintaining a record of the specification/design options which were considered. Specific directions are also provided for recording information pertaining to trade-off studies and rationales for decisions. Problem and change logs are also required to be kept; the method does not provide specific directions for recording these types of records.

Documentation Formats (Table 16)

A number of documents are required, with the formats specified by the method as either fixed or tailorable. Generation of a few documents is automated.

3.16.5 EASE OF USE

Technology Insertion

The developer provided the following estimates of minimum qualifications needed by a development team leader for successful use of the method: two to three years of college-level technical education, three to five years of development experience, working knowledge of one programming language, and experience on one software system. Understanding of the concepts of business economy, information systems theory, and structured programming was regarded necessary as well.

Training assistance includes hands-on demonstrations, overview presentations, classroom tutorials, and on-site consulting by the vendor, as well as a "hot line" service, user manuals, and a users' support group.

Learning time estimates were given as one day for a project manager to acquire an understanding of the major features and benefits of the method, three days of an experienced developer to learn to use the essentials of the method, and three months for such a developer to become an expert user.

Automated Facilities

The developer cited the tool "Graphdoc" as supporting the activities of change analysis, activity studies, and information analysis. This tool is provided by Epoc System AB (Sweden).

3.16.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

IBM PC-compatibles are appropriate for hosting the automated tools supporting the method.

Acquisition Costs

The single-user price to acquire the method and required components is $3200, and $2000 for a high-volume user. Technical training is $1000 per day, and a management overview is $1200 per day.

The method requires a training site license at $2000 per site.

Contact Information

Institute V
Box 6501
S-113 83 Stockholm
Sweden

+46-8-23 39 90

[provider]

3.16.7 REFERENCES

[Lund81] M. Lundeberg, G. Godkuhl, and A. Nilsson, Information Systems Development - A Systematic Approach. Englewood Cliffs, N.J: Prentice-Hall, Inc., 1981.

[Auer00] Auerbach Publishers, Inc., "The ISAC Approach to User-Oriented Systems Specification", in Systems Development Management, Information Systems Analysis: Analysis Methods and Tools.

3.17 **IStar** -- Integrated Project Support Environment

3.17.1 BACKGROUND

Synopsis

This method may be characterized as an integrated environment oriented towards project and data management approaches to software development. IStar provides a framework within which foreign tools, such as technical software development methods or language-specific compilers, may be integrated. Workbenches can be modified to take advantage of the method's user-interface and data-management facilities. In addition, IStar has its own tools for supporting activities inherent to project, data, and configuration management.

History

Imperial Software Technology of London undertook the development of IStar in 1983. It was first used commercially in 1986 and is available for general use. It varies from other support environments built from the bottom-up; instead of starting with language-specific tools, IStar began in its own development by defining the overall requirements for software project support and the associated database needs.

3.17.2 DESCRIPTION

IStar, Integrated Project Support Environment, supports the contractual approach to software development. In this approach, each task in a development project is viewed as a well-defined package of work that can be delegated to a contractor to perform for a client. It is up to the contractor to fulfill the contract within any constraints that may be specified. [Dows87]

The software development activities for which IStar prescribes detailed procedures are project and data management. The method assumes that the project organization is organized in a hierarchical manner, with the working relationships between people made explicit. The approaches upon which it is founded are entity-relationship modeling, binary-relationship modeling, and functional decomposition. There is no particular software process paradigm upon which the method is dependent for its most effective use, and the method is either well-suited or compatible with several important process models.

Within IStar there are a number of workbenches organized around activities regarded as necessary to any software development project. These workbenches, accessed via a common interface, provide support for project management, data and configuration management, technical development, Ada, quality management, and office automation/system administration. Initially, support was provided for three methods: CORE, VDM, and SDL.

Associated with each project task that is assigned by a client and accepted by a contractor is a contract database, which initially contains the client's specification of the work to be done. Over the course of the task, this database expands to record the progress of the work. The tools in IStar assist in managing multiple versions of the data items, keeping track of current status of the work, and sending completed deliverables to the client.

In addition to the contract database, there are work databases that are private to the user. These work databases contain user-selected workbenches appropriate for accomplishing the task at hand. These workbenches may be third-party tools which are compatible with the underlying operating system or tools provided within IStar; IStar provides guidance for integrating external tools. Results from the work databases, called transfer items, can be exported to the contract database. Within the contract database these transfer items are considered to be typed objects according to the tool used to create their contents. They also become part of the untyped configuration item that is used to contain the deliverable for a contract. For both transfer items and configuration items, IStar provides version control support.

3.17.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

The developer rated the method well-suited for use on applications involving embedded systems or process control, time critical or real-time processing, scientific or engineering processing, systems programming, and distributed processing or networks. Examples of delivered systems developed with the method include embedded defense, telecommunication systems, naval simulators, and a large-scale avionics project. It was estimated that between 5-20 organizations have used IStar on the same number of projects. The method is intended for medium and large-sized projects, and has been used on medium projects. The method is language-independent in the sense that workbenches can be provided for any language. However, the implementation languages which have been most frequently used for coding systems developed with IStar are C, Ada, FORTRAN, and assembler.

Target Constraints

The method does not address this aspect.

Modes of Expression (Tables 9,10)

A number of textual modes of representation are strongly encouraged by the method, including specified documentation templates, narrative overviews of modules, a formal specification language, and mathematical notation. Most iconographical modes of expression are compatible with the method; Booch diagrams are required and provided with automated support.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

Incremental or evolutionary development is required and rapid prototyping is strongly encouraged as means for clarifying system requirements. Change Control Board review is required and formal proof techniques, design reviews, and code walk-throughs are strongly encouraged.

Other Technical Aspects

The method supports automated requirements tools, which are seen by the developer as reducing efforts needed to incorporate changes in the requirements. It also supports, via the Genesis tool, Z and VDM proofs.

Assistance in ensuring consistency between specification, design, or code is provided via the problem reporting and change control tool part of the environment, via support for automated design tools, and via the configuration management system present. For Ada applications, the method's Booch diagram editor provides automated Ada code generation.

Reusability is addressed in terms of identification of reusable components by means of the library management tool and library contract tool.

3.17.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

IStar provides specific procedures and automated support for many activities associated with project management, including resource arbitration.

Communication Channels

The main way that the method is designed to facilitate communication among the parties involved in a software development project is through the contract model. This model provides support for tasks, data models, and roles. These roles correspond to project management activities and consist of project managers, resource managers, and cost managers. IStar provides tools to assist in the administrative and clerical tasks associated with these roles.

Quality Assurance (Tables 12,14,15)

Although IStar does not address quality assurance issues in particular, it can support the chosen quality assurance policy through checklists. These QA checklists can be passed to subcontractors, used by the client, or given to an independent QA contractor. QA checklists can form a hierarchy by referencing other checklists and can be kept in a checklist library. Their format is fixed and is produced from user responses to computer-directed prompts.

Automated recording procedures are provided for maintaining a record of specification/design options which were considered during the work effort, of the personnel who were involved in making a decision, and of all changes related to specificaiton/design decisions. Problem and change logs are provided in fixed format and generated from computer-directed prompts or automatically from data produced from other steps.

Documentation Formats (Table 16)

The method provides both fixed and tailorable formats for the required documents, all of which are provided with automated support. These documents are a quality assurance document, a user manual, and problem and change logs.

3.17.5 EASE OF USE

Technology Insertion

For successful use of the method, a development team leader would need as a minimum a bachelor's degree, three to five years of development experience, working knowledge of two programming languages, and experience on two different software systems. In addition, the concepts of structured programming and entity-relationship attribute modelling should be understood.

Training support includes hands-on demonstrations and overview presentations, classroom tutorials, on-site consulting by the vendor, an on-line help facility, "hot line" service, user manuals, a users' support group, and periodic technical updates. A project manager would need two to five days to acquire an understanding of the major features of the method. For an experienced developer, five to ten days would be required to learn the essentials of the method, and six months to achieve expert user level.

Automated Facilities

IStar itself provides a number of automated tools within itself. In addition, third-party tools support the activities of the method. The developer mentioned the following tools as offering specific support for the method and of interest to IStar users:

Name of tool	Tool vendor	Activities supported
Sun Trac	Sun	Project planning
Framemaker	Frame	Documentation
Interleaf		Documentation
Ada compiler	Alsys	Ada
Teamwork	Cadre	Analysis and design
Software through Pictures	IDE	Analysis and design
Genesis	IST	Formal method support

3.17.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

Configurations appropriate for hosting IStar include Sun Workstations, UNIX supported hardware, and Dec VAX/VMS.

Acquisition Costs

These costs are dependent on the licensing policy, which can be based on the number of users, or on a site, a multi-site, or a corporate license.

Costs to acquire the method and required components were given as $8,500 for a single user, $5,000 for a low-volume user, and $2,000 for a high-volume user. Acquisition costs for other recommended components supplied by the method vendor were not given, as they are particular to the number and type of tools. Training costs were $2,000 for low-volume users and $1,500 for high-volume users. A management overview costs $500.

Contact Information

Imperial Software Technololgy 01 581 8155
60 Albert Court
Prince Consort Road
London SW7 2BH
England [Provider]

3.17.7 REFERENCES

[Dows87] M. Dowson, "Integrated Project Support with IStar", IEEE Software, Vol. 4, No. 6, Nov. 1987, pp. 6-15.

[Dixo88] D. Dixon, "Integrated Support for Project Management", Proceedings of the 10th International Conference on Software Engineering, Singapore, April 11-15, 1988, pp. 49-58.

3.18 JSD -- Jackson System Development

3.18.1 BACKGROUND

Synopsis

This method employs the concept of a network of sequential processes which communicate with one another to address both the process and data aspects of requirements. The development of a system proceeds by composition, as opposed to decomposition. By precisely defining each successive increment to the system, the method emphasizes a system which at intermediate stages is well-defined albeit incomplete, as opposed to a system which though complete is imprecisely-defined.

History

First used for the development of a deliverable system in 1982, JSD is based upon JSP, or Jackson Structured Programming. JSD was developed in England by Michael A. Jackson, John R. Cameron, and colleagues.

3.18.2 DESCRIPTION

JSD, Jackson System Development, addresses itself specifically to activities involving system specification, system design, and system implementation. It is founded on object-oriented and event-oriented approaches to software development, and the concepts of process abstraction and structured programming are essential to the method. It is well-suited for use with rapid prototyping, the operational model, and the transformational model. The most effective use of the method is with the rapid prototyping or operational software process paradigms. As a result of using the method, parts of the coding activities can be skipped, since code can be generated from the products of earlier steps in the method.

JSD conceptualizes the problem facing a software developer in terms of two separate worlds: 1) the subject matter of the system, reflecting the "real world" entities and their behaviors, and 2) the implementation environment, consisting of the machinery and associated software facilities for running the completed system.

In constructing a model of the subject matter, the developer uses events, their attributes, and their orderings, to build a specification whose structure directly reflects the structure of the application domain. This specification differs from a "black-box" because JSD views the internal structure of the system as part of the problem statement, not a result of a particular implementation. The JSD model or specification thus elaborates the necessary data, processes, and communication streams of the real world, including semantics for timing requirements which may constrain the implementation of the system. This model is described as a network of communicating sequential processes, and represents an explicit simulation of the system's subject matter.

Theoretically, this specification is an executable form of the required software. However, in order to perform within the limited resources of the implementation environment, the system must usually be modified. This modification is accomplished through a mapping, the method providing transformation rules for this purpose.

Construction of the system specification is accomplished by following five steps in the method:

1) Entity/Action and Entity Structures Step;
2) Initial Model Step;

3) Interactive Functions Step;
4) Information Functions Step; and
5) System Timing Step.

In the first step, the real world is described at an abstract level in terms of entities which perform and suffer actions. Time-ordering is then considered when there are constraints in the sequence of actions, and other structures are added when it is necessary to describe an entity in terms of several sequential processes. The Initial Model Step then takes these structures of the first step and defines the set of sequential processes which are to model the problem domain. This is done by viewing each structure as the definition of the text of a particular type of process in the model. The addition of data stream or state vector communications enables modeling of the connections between processes to reflect changes that correspond to the dynamics of the real world. In the third and fourth steps, further executable operations and processes are added in order to produce the outputs of the system. These outputs may be messages to an external system user, in which case they are information functions, or functions which generate actions to be used in the model, in which case they are called interactive functions. The last step, or System Timing Step, specifies any constraints which would be necessary in the implementation to ensure reasonable performance. These include speed of execution and relative scheduling.

Implementation of the JSD model consists of transforming the network of sequential processes into a smaller set of programs which may be loaded and executed on the processors available in the implementation environment. Three of these transformation techniques include:

1) Process Scheduling;
2) State Vector Separation;
3) Process Dismemberment.

The procedure for achieving an implementation from the specification involves distributing the processes in the specification among the available processors or CPU's. The set of processes assigned to a single processor must then be combined into a single program, usually by means of a scheduler at the top level with a hierarchical structure of inverted process texts below. The scheduling algorithm for each processor must be defined, and a scheme devised for storing, accessing, and protecting the state vectors of the system. This may require the design of access logic to manipulate the data.

Thus, the method recognizes that the structure of the specification (which is in JSD an executable object) should not be the same as the structure of the implemented system. The former reflects the requirements, whereas the latter reflects the implementation environment.

3.18.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

The developer rated the method well-suited for applications involving embedded systems or process control, distributed processing or networks, data processing or databases, and large scale simulation or modeling. Specific types of applications for which delivered systems have been developed with JSD include a bank checking account system, software for a polyphonic music synthesizer, software for monitoring factory floor personnel/activity, and embedded software for a torpedo. There have been an estimated 21-100 delivered systems developed with the method, within 21-50 organizations. The method is intended for use on projects of all sizes; it has been used on small and medium-sized projects. Most frequently-used implementation languages for coding systems developed with JSD are COBOL, Assembly, and Ada.

Target Constraints

JSD addresses special features of the target hardware architecture with its rules for mapping the process network onto an arbitrary real/virtual processor configuration. Portability is assisted by these rules as well, the method's transformations allowing a given specification to be mapped onto different hardware configurations by using different process to processor mappings and different scheduling techniques. In addition, the method addresses concurrency issues by the fact that concurrency is a basic specification primitive of the method.

Modes of Expression (Tables 9,10)

A formal specification language is required; the method provides automated support for such. Required iconographical modes include process structures, process networks, and implementation diagrams, with automated support provided for the first two modes. Mapping rules are provided from process structures to process network, and from process network to implementation diagram. For facilitating transformation across phases of the software process, the method automates transformation through design and code at the prototyping level. For "production" software, the method provides rigorous transformation techniques which are partially automated.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

Executable specifications are required for clarifying system requirements or behavior, and several other techniques are encouraged. Design reviews are encouraged by the method.

Other Technical Aspects

Assistance in reducing efforts needed to incorporate changes in requirements is provided by automatic (re)generation of documentation and code via support tools, and a lack of redundancy in the technical documentation. Consistency is guaranteed between design and code at the prototyping level, because the design and code are generated from the specification; for "production" design and code, this consistency is partially guaranteed by automation. A data dictionary of fixed format is automatically generated by data produced from previous steps in the method.

The developer states that the method assists in the issue of reusability in the sense that code generated for prototyping can be re-used in the final production software.

3.18.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

A framework is provided in the method for assessing complexity; otherwise, JSD does not address management issues.

Communication Channels

Iconographic representations of the specification and design are the features of the method intended to facilitate communication within the development team, between the team and management, and between the development organization and the client. Prototypes are also used between management and the development team and with the client. The client is specifically involved in the software development process by review of the outputs of the modeling phase of the method, and review of the prototypes.

Quality Assurance (Tables 12,14,15)

Not addressed by the method.

Documentation Formats (Table 16)

A variety of formats and levels of automated support are provided for generating documents required to be produced.

3.18.5 **EASE OF USE**

Technology Insertion

Minimum qualifications needed by a development team leader for using the method successfully include two to three years of college-level technical education, three to five years of development experience, knowledge of one programming language, and experience on one software system. Although JSD training does not assume prior knowledge of any specific theoretical constructs, understanding of several topics (covered by the training) is required to use the method. These include entity/process structures, modeling with concurrent processes, and implementation techniques, such as scheduling/inversion.

Available training in the method includes hands-on demonstrations and overview presentations, classroom tutorials, on-site consulting by the vendor and others, user manuals, and a users' support group. It was estimated that a project manager would need five days to acquire an understanding of the major features and benefits of JSD. Ten days would be required for an experienced developer of five or more years' practice to learn to use the essentials of the method, and six months for an experienced developer to achieve the level of expert user.

Automated Facilities

The method provides within itself automated facilities for the representation modes required, as well as for several of the documents which the method requires be produced.

3.18.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

Configurations appropriate for hosting the method's tools include a PC or PS/2, IBM MVS, or DEC VMS systems.

Acquisition Costs

The costs to acquire the method are given in terms of volume of users, and consist of recommended components supplied by the vendor, as well as technical training and management overview. For recommended components, the unit costs are: for single-user, $10,000, for low-volume user, $5,000, and for high-volume user, $1000. Technical training costs run from $2,000 for a single user, to $1800 and $1000 for low and high volume users, respectively. Management overviews are $1000 for a single person, $900 for a few, and $500 for many.

The licensing policy varies with the configuration and volume. PC-based software is priced per machine, with price breaks at high volumes. Mainframe software is priced per site.

Contact Information

Michael Jackson Systems Ltd.
22 Little Portland Street
London WIN 5AF
United Kingdom

(01) 499 6655

[Provider]

3.18.7 REFERENCES

[McNe86] A. T. McNeile, "Jackson System Development", in Information Systems Design Methodologies: Improving the Practice, T. W. Olle, H. G. Sol and A. A. Verrijn-Stuart, eds., Elsevier Science Publishers B. V. (North-Holland), IFIP, 1986.

[Came86] J. R. Cameron, "An Overview of JSD", IEEE Transactions on Software Engineering, Vol. SE-12, No. 2, Feb. 1986.

3.19 MASCOT -- Modular Approach to Software Construction, Operation and Test

3.19.1 BACKGROUND

Synopsis

Mascot, from the beginning of the software development process, addresses concurrency, distribution and real-time behavior, and provides a means of discussing or reasoning about feasibility. This is in contrast to other similar methods, which bring up these concerns late in the process. Mascot also provides explicit directions and procedures for system specification, system design, system implementation, and software quality assurance. The method is available for general use.

History

Mascot started out in the United Kingdom's research and development establishment, RSRE, Malven in 1971. Its principal architects were Ken Jackson, formerly of the Ministry of Defense, and Hugo Simpson, formerly of the Royal Air Force. Mascot2 has been in use since 1976 and Mascot3 since 1983.

3.19.2 DESCRIPTION

Mascot, Modular Approach to Software Construction, Operation and Test, addresses system specification, design, implementation, and software quality assurance. It is founded upon a functional decomposition approach, as well as on data flow-oriented, data structure-oriented, and object-oriented approaches. Its use fits into the context of a variety of software process paradigms, but none of these paradigms is strongly linked to the most effective use of the method. Essential to the method are the concepts of information hiding, abstract data-types, genericity, and module coupling/cohesion. Additionally, Mascot defines design architecture to be identical to implementation architecture, and therefore steps involving any transformations between these two representations in certain process models can be skipped.

The starting point of the method is a "Design Proposal" which identifies the major functional subsystems and Top Level Internal Data Stores. The design progresses by decomposing the top level units into lower level components identifying active, passive and device dependent components. The networks are specified in terms of data flow and functionality with connectivity between components defined in terms of Access Interfaces. The network decomposition terminates when "simple" elements have been identified which can be directly implemented in a programming language. The complete design architecture is checked for self-consistency by a process known as "status progression". This procedure ensures that the design structure is sound before implementation can proceed. The complete system can only achieve fully-checked status when all modules depended upon have achieved a similar status.

The specification of Simple Element Templates can make use of techniques such as State Transition Diagrams, Finite State Automata, PDLs, structured text, and so on. The "Intercommunication Data Areas" (IDAs) make use of the special "kernel primitives" to achieve the required characteristics in terms of behavior and integrity. Testing starts with these IDAs and then proceeds by gradually producing more complete test networks. All the test networks use exactly the same architectural construction constraints as the main system development. Thus the testing phase provides the first "re-use" (or perhaps it should be "pre-use") of the templates developed for the main system.

3.19.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

The developer states that Mascot is well-suited for applications involving embedded systems or process control. time critical or real-time processing, distributed processing or networks, and large scale simulation or modeling. Delivered systems developed using the method include guided missile systems, avionics systems, military communications systems, process control (e.g. beer manufacturing), general telecommunications, and traffic control (air and road). In all, more than 250 systems have been delivered that were developed with the method, within between 21 to 50 organizations.

While the method is intended for medium to large systems, it has been used for projects of all sizes. Implementation languages most frequently used in conjunction with the method's use are Coral66 (the United Kingdom's military language), C, Pascal, and RTL 2.

Target Constraints

Mascot prescribes steps for handling several target system requirements. The behavioral model, which encompasses concurrency, process synchronization, and deterministic scheduling via the kernel (the method's own run-time system), enables timing constraints to be accomodated. Spatial constraints are addressed by identifying all Internal Data Stores and enabling them to be placed in defined locations. The method specifically supports multi-processor distributed architecture; automated support includes on-line debugging of such distributed systems. Special features of the target operating system are addressed by Mascot's defining its own run-time system ("tokens") which can be built on a bare target. In addition, Mascot provides the means for creating an application-specific set of "operating system-like" facilities.

By specifically distinguishing between concurrent (active) processes and passive data objects in the design representation, the method can address concurrency issues. There are also guidelines for assessing fall-back modes to handle fault-tolerance issues. The developer states that, because the design architecture is independent to a large extent from the target configuration, the method assists in porting systems to different target configurations. Specific placement can be applied without changing the design, or, if necessary, by changing specific "Intercommunication Data Areas" from single processor versions to distributed versions, and by altering code in device-dependent "server" modules.

Modes of Expression (Tables 9,10)

The method requires narrative overviews of modules as well as Mascot-specified diagrams, for which automated support is provided. Required iconographical modes include data-flow diagrams and Mascot-specific hierarchy charts; automated support is provided for both of these modes of representation.

The method is seen to facilitate transformation of representations across phases of the software process by making identical the design architecture and the implementation architecture, thus eliminating the need for a transformation.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

Incremental or evolutionary development is required, as are data-flow analysis and design reviews. Data-structure analysis is encouraged for stored "objects".

Other Technical Aspects

Mascot emphasizes well-defined interfaces and general architectural features in order to confine area(s) of change within the design, thus facilitating the incorporation of changes in the requirements. By enforcing the introduction of change at the design level, and "rippling down" the change into the implemen'ation, the method assists in ensuring consistency between the design structure and the implementation structure.

The concept of reuse is integral to the method, in that the design architecture is totally predicated on the notion that all "components" are derived from "templates". Thus, all templates are by their nature re-usable and are always specified in terms of the Access Interfaces they provide or require (using Windows and Ports).

3.19.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

Guidelines are provided for a number of project management activities, and specific procedures are incorporated for estimating initial cost and reliability, as well as for tracking project progress, for which the method provides automated support.

Communication Channels

Mascot facilitates communication among members of the technical team by providing a diagram notation which gives a clear view of the system to be developed. This view includes identification of all interfaces, distinguishing between active (concurrent processes) and passive data objects, a dynamic behavioral model, and representation of placement of components in distributed hardware targets.

Communication with the client is facilitated by requiring the developer to identify required functions and required interactions with the environment. The design enables traceability to be established back to these items. The client is involved by having him sign off on these required functions and interactions at the beginning of the design stage. He is also required to approve acceptance test procedures.

Quality Assurance (Tables 12,14,15)

Mascot provides specific procedures for test planning at one or more points in the software process, generation of tests based on system requirements, and unit/integration testing. Early detection of inconsistency and/or errors is assisted by the "status progression" process, mentioned in the Description section above. Also provided are specific directions for recording and maintaining information regarding the rationale for any decision made during the software development process.

Documentation Formats (Table 15)

A number of documents are required to be produced. The prescriptiveness of the format varies between the documents. For more specific information on the degree of prescription, the tailorability and the degree of automation, see the associated table.

3.19.5 EASE OF USE

Technology Insertion

The developer estimated that a team leader would need less than two years of college-level technical education, one to two years of development experience, knowledge of one programming language, and experience on two different software systems in order to successfully use the method. The concepts of asynchronous concurrency and data-flow analysis should be understood as well.

Training assistance included overview presentations, classroom tutorials, on-site consulting, a "hot-line" service, a users' support group, and periodic technical updates. One day would be required for a project manager to understand the major features and benefits of the method, while an experienced developer could learn to use the essentials in five days and become an expert user in two to four months.

Automated Facilities

The developer mentioned two tools that support diagram drawing: ECLIPSE, from Software Sciences, and VSF from Systematica. In addition, the method provides automatically generated system structure charts and automated support for tracking project progress.

3.19.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

The tools supporting the method are hosted on VAX/VMS.

Acquisition Costs

Technical training costs $1500 and a management overview, $300. These costs are set regardless of the number of trainees. Product costs were not provided; however, the applicable licensing policy is per CPU.

Contact Information

Ken Jackson
SD-Scicon plc.
Pembroke House, Pembroke Broadway
Camberley, Surrey GU15 3XD
England [Provider]

3.19.7 REFERENCES

[DRIC00] Defence Research Information Centre, "Handbook of Mascot 3.1", Glasgow, Scotland.

[SEJ 86] Software Engineering Journal, May 1986: special edition on Mascot3 containing four papers covering design representation, the method, use with Ada, and testing. Jointly published by IEE/BCS in London.

3.20 MBOOD -- Model-Based Object-Oriented Design

3.20.1 BACKGROUND

Synopsis

This method provides a strategy for identifying objects from a model of the system to be built. This model, based on the requirements of the system, is composed of four major components: the entity model, the data model, the behavior model, and the use model. A data dictionary provides definitions for terms used in any of these component models.

History

MBOOD was developed by David M. Bulman and Erin Bulman. It was first used for the development of a deliverable system in 1985. The requirements analysis and preliminary design phases incorporated by the method are derived from a combination of DeMarco's Structured Analysis, McMenamin and Palmer's Essential Systems Analysis, Hatley and Pirbhai's Real-Time Analysis, all joined together with conceptual modelling approaches, using a form of entity-relationship diagrams.

3.20.2 DESCRIPTION

MBOOD, Model-Based Object-Oriented Design, prescribes specific procedures for conducting activities involving requirements definition, system specification, and system design. In the requirements analysis phase, a model of the problem requirements in the problem domain is built. This model is an extended form of a structured specification and includes four components: 1) the entity model, an informal representation of how the user views his world in terms of objects and their relationships; 2) the data model, including data flow diagrams and entity-relationship diagrams; 3) the behavior model, consisting of process specifications which show responses of the system to events, and control specifications in the form of state transition diagrams; and 4) the use model, which consists of the use case list and the services list [Jaco87] (Author's note: see the description of ObjectOry for an explanation of these terms).

In the preliminary design phase the method also uses the ideas of McMenamin and Palmer's Essential Systems Analysis in distinguishing between the essence of the problem and an incarnation of a solution. The first part of the design is formed by constructing the incarnation of a solution, which takes into account all constraints concerning hardware to be used, for example.

The information recorded in the previous phases is used to derive a list of candidate objects in the object design phase. Each of the objects on this list is specified with a set of operations on the object; it is critical to identify and completely specify the operations on an object, because objects consist of private data, together with operations on that data. These operations are each specified with a name and parameters, and each parameter is given a type and declared as in, out, or in-out.

The object candidate list comes from all parts of the structured specification. When using the data flow diagrams, for example, most objects are derived from the data stores, terminators, and some kinds of data flows. The operations are determined by a technique called "carving" from the processes connected to the candidate objects. The method also includes details for deriving objects from several other sources including entity-relationship diagrams, and state transition diagrams. Additional objects are identified as resources for those candidate objects already listed. This step includes the identification of abstract data type objects.

Next, two different kinds of refining are done to the objects in the candidate list. First, object candidates may be split into two or more objects, or some of them may be combined into one object. This is normally done either to simplify the interfaces to objects, or to make them more reusable. Second, the specification of individual objects is revised: operations may be added to make the object (more) complete, or higher-level operations, such as iterators, may be substituted. For each different kind of object in the taxonomy, a check-list of "proto-operations" is used to help make objects more reusable.

Finally, the method addresses system architecture. Once the objects are defined and refined, they are organized into a system. The method includes two approaches: one shows how to insert the objects into a known architecture, e.g., a structured design. Alternatively, given a development environment that allows it, the method shows how to construct an object-oriented architecture.

3.20.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

The developer rated the method well-suited for a number of application areas (see Table 5) and reported that delivered systems have been developed using the method for real-time data acquisition applications, flight simulators, weapons control, and an on-line financial system. In all, the method is estimated to have been used with between five to twenty delivered systems, within between 21 to 50 organizations. The method is intended for use with projects of all sizes; it has been used on small and medium-sized projects. Ada and Objective-C were listed as the implementation languages most frequently used when coding systems developed with MBOOD.

Target Constraints

The method can address special features of the target hardware architecture by showing, when applicable, how to encapsulate such features, e.g., making special I/O devices into individual objects. Likewise, the method shows how to derive objects which hide or encapsulate special or troublesome features of a target operating system. The result, when changing to a different hardware target, is that only the internals of such objects need to be changed. The operations of the objects remain unchanged, and thus the rest of the code which uses those objects need not be changed, thereby assisting in portability.

The method also gives detailed guidelines for determining which objects should be concurrent and how such objects must communicate. Fault-tolerance issues are addressed in a minor way in the identification and use of "daemon" objects.

Modes of Expression (Tables 9,10)

The method strongly encourages using decision tables, finite-state diagrams, data-flow diagrams, control-flow diagrams, and entity-relationship diagrams. Warnier/Orr diagrams and flowcharts were considered inconsistent with the method.

The method prescribes mapping rules for translating from one more to another. Specifically, rules are prescribed for transforming data-flow diagrams into object specifications and inter-object dependencies.

Across phases of the software process, the original purpose of the method was to show in detail how to derive the detailed specification of the objects of an object-oriented design from a structured specification. More recent versions of the method also show how to derive objects from only a partial structured specification. The derived objects form the basis for writing the Ada package specifications and task specifications. This Ada code is merely a simple transliteration of the object specifications derived in the design phase.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

A number of analysis and review techniques are strongly encouraged by the method. See Table 12 for details.

Other Technical Aspects

The method assists in identifying possible reusable components in the following way. Most of the objects derived by the method are one of two different kinds. The first are those which represent useful entities in the problem domain, or devices the system must deal with. The representation is done by encapsulating the data describing the entity, together with all the operations which can be performed on that data. The second kind are abstract data type objects, which encapsulate a data type from a problem domain, together with the operations on entities of that type. Both kinds of objects tend to be reusable in any programs which deal with the same problem domain.

3.20.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

The method provides guidelines for assessing complexity; otherwise, it does not address activities associated with project management.

Communication Channels

All parts of the requirements analysis and preliminary design phases are designed to facilitate and coordinate communication between all parties involved in the software project; additionally, all parts of the object design phase are intended to facilitate communication within the development team.

Quality Assurance (Tables 12,14,15)

MBOOD provides guidelines for prescriptive checking of interfaces; unit/integration testing is required but the method does not provide directions for accomplishing such testing. The developer stated that by incorporating a full, and augmented, Structured Analysis as the first step, many kinds of errors are detected early. The combination of all parts of the augmented Structured Specification (e.g., data dictionary, data/control flow diagrams, control specifications in the form of state machines, entity-relationship diagrams, and event-entity matricies) was seen as assisting with detection of most specification errors (both inconsistencies and ambiguities) at an early stage.

Documentation Formats (Table 16)

A tailorable format is provided for most of the documents required by the method; see Table 16 for details.

3.20.5 EASE OF USE

Technology Insertion

The developer offered the following estimates of the minimum qualifications needed by a development team leader for successful use of the method: less than two years of college-level technical education, three to five years of development experience, knowledge of one programming language, and experience with one software system. An understanding of finite-state machines would be of help with problems with complex control requirements.

Classroom tutorials and on-site consulting by the vendor or independent consultants are available to train an organization in the use of the method. It was estimated that one day would be required for a project manager to acquire an understanding of the major features of the method. An experienced developer would need eight days to learn to use the method's essentials, and two months to become an expert user.

Automated Facilities

The developer reported that most CASE vendors support the basic needs of the method.

3.20.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

Not applicable.

Acquisition Costs

The applicable costs relate to technical training. For a single user, unit training costs are $1,200. For a low-volume user, the unit costs are $600, and for high-volume users, $500.

Contact Information

Pragmatics, Inc. 808-883-9011
P.O. Box 3429
68-1824 Pau Nani Street
Waikoloa, HI 96743 [Provider]

3.20.7 REFERENCES

[Bulm88] D. M. Bulman, unpublished manuscript submitted for publication.

3.21 MINI-ASYST -- The MINI-ASYST Development Methodology

3.21.1 BACKGROUND

Synopsis

MINI-ASYST provides a framework for development of information systems which covers the entire development process from user needs identification through implementation and maintenance. It is intended to provide a closely defined process for development of small to large size systems.

History

MINI-ASYST is geared toward information systems development, and is based upon ASYST-Development, a method developed in 1980-1981 by Atkinson Tremblay & Associates. ASYST-Development is based upon structured techniques of analysis and design, incorporating data and systems modeling techniques. MINI-ASYST was first used with respect to a deliverable system in 1986.

3.21.2 DESCRIPTION

MINI-ASYST is a method which provides a well-defined framework for developing information systems using structured techniques. The method covers the entire software development process, partitioning this process into five separate phases, namely, (1) mandate definition, (2) feasibility study, (3) design and specifications, (4) build and test, and (5) implementation. The method specifies for each phase the activities needed to carry out that phase. Specific methods are proposed for dealing with the important features of each of the development phases. Within this framework, it is possible to tailor the method to individual needs of the development organization, and of the particular application.

In the MINI-ASYST method, analysis and design aspects are treated concurrently during the development phases. The technical feasibility of each element of the proposed functional solution must be established, but internal details of the solution are not described. The systems architecture is developed in three stages: a global architecture, a detailed analysis of the system's functions and technical aspects, and a final architectural review to adjust and finalize the system before it is built.

In conjunction with development, MINI-ASYST provides project management assistance, including allocation of resources to activities, use of estimation techniques, and steps for monitoring phase progress. For each phase, the method specifies what steps are necessary to ensure completeness, what activities should be occurring during the phase, and what deliverables, including documentation, are to result.

3.21.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

The developer considered the method to be compatible with applications involving systems programming, distributed processing or networks, data processing or database, and large scale simulation or modeling. The developer rated the method inappropriate for use in applications involving embedded systems or process control, time-critical or real-time processing, or scientific/engineering systems. The method has been used to develop small to very large business systems in different industries, including government, utilities, manufacturing, and insurance. It is intended for use on projects of all sizes, and has been used as such, on an estimated 21-100 systems developed within 51-100 organizations. While the method supports all languages, most of the systems developed are business systems with COBOL as the language most frequently used. The second most-frequently-used language is ORACLE.

Target Constraints

Timing and spatial constraints are handled through support of Operational Procedure Diagrams. Special features of the target hardware architecture and operating system are addressed with the definition of four architectures: 1) data, 2) systems, 3) technology (hardwa·e and software), and 4) I.S. organization (people architecture). Security of access is addressed by definition ol data access, site access, and people access.

Although MINI-ASYST is totally independent of the target configuration, during Phase 2 (feasibility study) the recommended alternative is proposed and planned in light of the chosen hardware and software.

Modes of Expression (Tables 9,10)

The method strongly encourages specified documentation templates and narrative overviews of modules as textual representations. Strongly encouraged iconographical modes are data-flow diagrams, entity-relationship diagrams, and flowcharts. Finite-state diagrams are considered to be inconsistent with the method.

The method prescribes mapping rules for translating from data-flow diagrams to structure charts, and from entity-relationship diagrams to logical data structure. The transformation across the phases of the software process is facilitated through iteration and refinement using data and systems modeling at the high level model, the structure charts, and pseudo code.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

A number of techniques are strongly encouraged both for requirements clarification and for analysis and review; see Tables 11 and 12.

Other Technical Aspects

The method facilitates incorporation of changes in the requirements by a log of changes that are considered through the use of structured techniques. Ensuring consistency between entities such as specification, design or code is addressed with techniques within the method.

With respect to identification of possible reusable components, the whole approach of MINI-ASYST is based on re-using what are called "Development Components"; these can be any information produced during the development process.

3.21.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

The method provides guidelines for performing all the activities shown in Table 13, with the exception of Pert or Gantt chart generation. In addition, it provides a framework for producing work breakdown structures.

Communication Channels

Within the development team, communication is facilitated by the One Page Concept: the whole method stands on a 3-ft. x 4-ft. poster intended to keep the whole process simple.

This One Page Concept is also seen as facilitating communication between the technical development team and management, as well as between the development organization and the software client. The poster clearly identifies the role of the project manager versus the systems developers, as well as the role of the outside players. The method indicates for each phase of the development where the software client should be involved. Moreover, throughout the development process, the method recommends structured walk-throughs involving the client.

Quality Assurance (Tables 12,14,15)

MINI-ASYST provides a framework for accomodating various testing activities. It requires that records of technical decision-making be kept, but does not provide specific directions for recording information.

Documentation Formats (Table 16)

A number of documents are required by the method; most of them can be tailored by the using organization. See Table 16 for specifics.

3.21.5 EASE OF USE

Technology Insertion

Minimum qualifications needed by a development team leader for successful use of the method were estimated as two to three years of college-level technical education, one to two years of development experience, knowledge of one programming language, and experience with one software system.

The developer estimated that a project manager would need two days to acquire an understanding of the major features of the method. An experienced developer would need ten days to learn to use the method's essentials, and three months to achieve expert user level.

Training is available in the form of hands-on demonstrations, overview presentations, classroom tutorials, on-site consulting by the vendor and independent consultants, a "hot-line" service, user manuals, and periodic technical updates.

Automated Facilities

The method is supported by the DEVELOPER, a front-end CASE tool. However, the method and the DEVELOPER are sold completely separately.

3.21.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

For further information on the DEVELOPER support tool, contact the provider of MINI-ASYST.

Acquisition Costs

Cost to acquire the method for a high-volume user was given as $25,000. Technical training for low-volume users is $5,000. MINI-ASYST is sold under a corporate license.

Contact Information

ASYST Technologies, Inc.
1080 Beaver Hall Hill, Suite 1400
Montreal, Quebec
Canada H2Z 1S8

514-871-0108
1-800-361-3673 (U.S. only)

[Developer & Provider]

3.21.7 REFERENCES

Contact the developer for further information.

3.22 MULTI/CAM -- MULTI/CAM - SDM/STRUCTURED

3.22.1 BACKGROUND

Synopsis

This method may be described as a workstation environment which integrates internal and external tools into a unified development system. The interface between these tools makes moving between activities transparent to the user. The approach to development is based on Structured Analysis and Design.

History

Initial system development and introduction of the method occurred in 1976, by AGS Management Systems, Inc., the developer of the method; the current updated system was first used in 1986 for the development of a deliverable system. The method is an extension of SDM/STRUCTURED, from the same company, which is an aid for constructing information systems.

3.22.2 DESCRIPTION

MULTI/CAM operates as a workstation with integrated software development and project management tools. The developer states that the method prescribes specific directions and procedures for all development activities shown in Table 1, as well as for building purchased packages and small projects. In conjunction with the framework of SDM/STRUCTURED, MULTI/CAM provides support for the development of new systems, as well as for the enhancement and maintenance of existing systems.

SDM/STRUCTURED is the approach to software development provided within MULTI/CAM. A project is initiated by a Service Request made by the software client. The framework provided by SDM/STRUCTURED consists of several phases covering the work to be done on a project:

1. Project Valuation Assessment - a document assessing the potential project for risk factors and cost effectiveness;

2. System Requirements Definition - the proposed system's requirements are stated along with objectives and basic problems which the system is intended to overcome;

3. System Design Alternatives - different solutions to the problem are evaluated;

4. System External Specifications - a specification of the system's operation, understandable to the software client, is prepared, in order to obtain the client's approval.

5. System Internal Specifications - this phase develops the design of the system such that the external specification is satisfied within the technical constraints.

6. Program Development - code is written according to the system design specifications.

7. Testing - this activity spans several phases and involves test planning, test specification, and executions.

9. Implementation - includes all activities supporting the transition to the new system, e.g., training, installation.

10. Post Implementation Review - the new system is evaluated following the first six months of use of the system.

3.22.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

The developer reported the method well-suited for use with embedded systems or process control, systems programming, distributed processing or networks, data processing or database, and large scale simulation or modeling. Examples of delivered systems built with the method are financial information systems, project management systems, and inventory control systems. The method is intended for and has been used on projects of all sizes, amounting to more than 250 delivered systems developed with the method, within more than 100 organizations. Implementation languages most frequently used when coding these systems are COBOL, FORTRAN, Ada, Pascal, and C.

Target Constraints

The developer stated that the method prescribes steps for handling all the requirements of the target system as shown in Table 8, with the exception of fault-tolerance issues. Further information detailing how the method addresses these constraints was not provided.

Modes of Expression (Tables 9,10)

Required textual modes are specified documentation templates, narrative overviews of modules, and structured English. Data-flow diagrams are the required iconographical representation. The method prescribes mapping rules for translating from data-flow diagrams to structure charts. Across phases of the software process, the method facilitates transformation through detailed summarization of the phase work into a format suitable for use by the next phase.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

The method encourages a number of techniques for requirements clarification; see Table 11. Data-flow analysis is required, and several other analysis and review techniques are strongly encouraged; see Table 12.

Other Technical Aspects

The developer stated that the method assists in reducing the effort needed to incorporate changes through complete change control guidelines and control worksheets. These procedures are reported as helping to ensure consistency between specification, design, and code.

Through segregation of the code functions, the developer states that the method assists in identifying possible reusable components.

3.22.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

The method emphasizes support for project management and specifically addresses a number of such activities. See Table 13 for further information.

Communication Channels

Specific features of the method designed to facilitate communication within the development team include the definition of roles and responsibilities, as well as reviews and use of sign-off documents at the conclusion of each phase. In addition, each team member can communicate with other team members through the mail facility and can access a centralized documentation repository for review at any time.

Between the technical development team and management, there are defined administrative functions. Key checkpoints, roles, responsibilities and decision points are highlighted to give the project leader and technical development team the guidelines needed to manage the life cycle process.

To facilitate communication between the client and the development organization, the method provides procedures for performing an evaluation of the system proposed by the client. This evaluation considers benefits, staffing, economic considerations, system scope, and justification. The client takes an active role in the life-cycle process, helping to determine requirements during detailed interviews, and meeting with the development organization throughout the process to review and sign-off on progress.

Quality Assurance (Tables 12,14,15)

The developer reported that the method provides specific procedures for test activities, as shown in Table 15, and furnishes automated support for test planning. Relevant documents include internal program documentation and a quality assurance/test plan. The method strongly encourages design reviews, code walk-throughs, Facilitated Application Specification Techniques, and Change Control Board reviews.

Automated recording procedures are provided for maintaining a traceable record of technical decision-making during the software development process. Documents automatically generated are a design decision log, a problem log, and a change log. For further detail, see Table 14.

The method provides specific procedures for configuration management.

Documentation Formats (Table 16)

All documents required by the method are tailorable within the method and generated automatically based on data produced from other step(s) in the method. See Table 16 for specific documents.

3.22.5 EASE OF USE

Technology Insertion

The developer estimated that the minimum qualifications for successful use of the method by a development team leader were two to three years of college-level technical education, three to five years of development experience, knowledge of one programming language, and experience with three to four different software systems. Major theoretical concepts which should be understood by an experienced developer are Structured Analysis and Design.

Training assistance is provided in the form of hands-on demonstrations, overview presentations, classroom tutorials, on-site consulting by the vendor or independent consultants, on-line tutorials, on-line help facility, a "hot- line" service, user manuals, a users' support group, related publications from third-parties, and periodic technical updates.

The number of days required for a project manager to acquire an understanding of the major features and benefits of the method were estimated as five. An experienced developer would require two days to learn to use the essentials of the method. One month would be required for an experienced developer to achieve the level of expert user.

Automated Facilities

The developer reported that MULTI/CAM supports the use of all CASE/IPSE tools, such as 4GL's, diagramming, code generators, project management packages, reverse engineering activities, prototyping, and business planning. MULTI/CAM can be customized to support a company's preferred software products.

MULTI/CAM also provides automated facilities for several areas of software development. See Tables 13, 14, 15, and 16.

3.22.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

An IBM PC XT or AT with PS/2 and 20 MB hard disk is appropriate for hosting MULTI/CAM.

Acquisition Costs

Cost to acquire the method and required components is $4,000 for a single user. For a hign-volume user, the cost is $97,000. Cost for technical training or management overviews is $850.

There is a perpetual site license for the system.

MULTI/CAM

Contact Information

AGS Management Systems, Inc. 215-265-1550
880 First Avenue
King of Prussia, PA 19406 [Developer & Provider]

3.22.7 REFERENCES

Further product information is available from the developer.

3.23 ObjectOry

3.23.1 BACKGROUND

Synopsis

This method is an object-oriented technique for developing large systems. Comprising the method are three independently-developed techniques: block design, conceptual modeling, and object-oriented programming. All three of these techniques incorporate the major characteristics of object-orientation: data abstraction, information hiding, dynamic binding and inheritance. The method is available for general use.

History

ObjectOry was developed by Ivar Jacobson and was first used in 1968 for the development of a deliverable system. It is related to the AXE method, developed in 1969-1972. The design framework used by the method, called block design, originated within Ericsson Telecom, a telephone company in Sweden. ObjectOry is basically the paradigm behind the CCITT SDL, or Specification and Description Language.

3.23.2 DESCRIPTION

ObjectOry specifically addresses activities in the software process involving requirements definition/clarification, system specification, system design, system implementation/installation, and software quality assurance. It is founded on object-oriented, service-oriented, and user-oriented approaches. Regarding software process paradigms, the most effective use of the method is in the context of the factory model, where a factory encompasses *both activities and objects manipulated by the activities,* and where *subfactories* can be defined at lower levels of detail. Concepts essential to the method are information hiding, process abstraction, abstract data-types, inheritance, and module coupling/cohesion.

The method makes a distinction between blocks, or application modules, and components, or reusable program elements. A system is built by means of creating application-specific blocks and combining them with other blocks which already exist. Blocks may be made up of lower-level blocks or of components. At the lowest level, blocks and components are implemented as classes in an object-oriented programming language.

System development is conceived as a factory with two sub-factories, System Analysis and System Design. Within System Analysis are three sub- factories: entities modeling, use cases modeling, and services modeling. Entity Modeling describes real world objects and the relations between those objects from a static viewpoint. Use cases and services are extensions provided by the method to allow for dynamic behavior modeling.

Use cases are behaviorally related sequences that describe different aspects of the system, which is regarded at this level as a black box. People who participate in the operation of a system are called users, e.g., telephone subscriber or bank clerk, and these users, in a dialog with the system, can perform a series of transactions, or use cases. The collection of use cases is represented in a conceptual diagram.

Related to the concept of use cases is the concept of services. This concept is introduced in order to assist in modularizing the system, not easily done with use cases themselves. Services are clusters of the parts which are similar among several use cases. Not only does the concept of services provide a means for identifying packets of the system that contain behaviorally related functions, but also such a concept is useful for providing ordering units when offering the system to a client. Services are seen by the client as indivisible packets of functions, which functions the client either wants in total or wants not at all.

Input to System Design consists of a system specification with conceptual diagrams representing entities, use cases, and services. The three sub- factories of System Design are System Level Design, Block Design, and Component Level Design. Each of these sub-factories contain sub-factories of its own.

An ideal structure for a system would consist of implementing as a block each entity and each service that were defined during System Analysis. However, performance requirements and other behavioral requirements will also influence the structure. The design is represented as blocks and communication paths between the blocks. Related activities also occur involving further implementation of use cases and use case testing. The method also provides guidance in concepts related to reusability, such as classification of blocks for use in adaptation of a system for a customer, and incorporation of components from a library or planning new components which would be useful in many other blocks.

3.23.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

The developer rated the method well-suited for a number of application areas, including 3-D graphical presentation, compiler design, and CASE development. Specific types of applications for which delivered systems have been developed using the method include large systems with long life-times, e.g., telecommunication systems and automatic inventory systems. Estimated numbers of delivered systems using this method were given as between five and twenty, within the same number of organizations. Although the method is intended for medium-sized projects, it has been used to build both medium and large systems. Most frequently used implementations langu.ges have been Smalltalk-80, Objective C, and PLEX, a programming language developed by Ericcson Telecom.

Target Constraints

The developer stated that constraints are handled by applying specific transformation rules. These rules depend on the application and implementation environment. The method addresses portability issues via language independent design, or blocks, as well as via inheritance, which puts changes in one place.

Modes of Expression (Tables 9,10)

The method requires a number of textual modes of representation, all of which are provided with automated support. Iconographical modes of representation required and provided with automated support are finite-state, entity, use-case, service, interface, and block diagrams. Hierarchy charts are required but not automated.

The method prescribes mapping rules for translating between use case model to service model, from entity and service model to block model, from use case model to interface model, and from block model to programming language.

Between specification and design, analysis models are mapped to blocks in a seamless way. Between design and code, translation is facilitated by expressing the design in communicating blocks. Blocks are implemented with code and components, which process is guided by a set of implementation dependent rules.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

The method utilizes several analysis and review techniques. These include entity analysis, use-case, service analysis, interface analysis, and Change Control Board review, all of which are required. Several other techniques are encouraged which would clarify system requirements or behavior.

Other Technical Aspects

Change in requirements is considered a normal process, and the solution is structured according to this concept. System development is viewed "as an activity that changes a system from being one 'thing' to being another different 'thing'. The first development cycle is only a special case and a change from 'nothing' to 'something'" [Jaco86]. One specific technique used by the method to facilitate change is inheritance, which is also the way that reusable components are identified. The method assists in ensuring consistency between its specification models, design models, and programming models by the principle of "seamlessness". Two models may be said to be seamlessly related to one another if concepts introduced in one of the models can be found in the other model through a simple mapping [Jaco87].

3.23.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

ObjectOry does not address project management.

Communication Channels

The concepts which are designed to facilitate communication within the development team are the factory concept, use cases, services, blocks, components, and entities. Between management and the development team, the factory concept, use cases and services are used to facilitate communication. The client and development organization communicate in terms of entities, use cases and services. In addition, all analysis models can be used to involve the client, early in the analysis phase.

Quality Assurance (Tables 12,14,15)

The method prescribes specific directions and supports with automated procedures test planning, test generation based on system requirements, unit/integration testing, field/acceptance testing, and testing oriented towards use cases. Use cases are expressed in operations on entities. If an operation is missing, it implies inconsistency. Design reviews and code walk- throughs are encouraged, and several analysis techniques are required.

Automated recording procedures are provided by the method for maintaining a record of specification/design options which were considered, trade-off studies, rationale for any decision, personnel involved in making a decision, and all changes related to specification/design decisions. An object dictionary and change log are automatically generated from other steps and follow a format fixed by the method.

Documentation Formats (Table 16)

All of the documents required to be produced by the method feature automated support. Tailorability and level of automated support vary between documents.

3.23.5 EASE OF USE

Technology Insertion

The developer estimated the following as minimum qualifications a team leader should have in order to successfully use the method: a bachelor's or advanced degree (depending on the application), three to five years of development experience, working knowledge of two programming languages, and experience on one software system. Concepts needed for successful use are semantic networks and finite-state machines.

The developer provides training assistance in the form of overview presentations, classroom tutorials, on-site consulting, and user manuals. It was estimated that a project manager would be able to acquire an understanding of the major features and benefits of the method in one day. Experienced developers would need between ten and twenty days to learn to use the method's essentials, and four to six months to become expert users.

Automated Facilities

Specific information on tools or environments supporting the method was not provided.

3.23.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

This information was not provided.

Acquisition Costs

No information was provided as to cost; however, there is a licensing policy which is negotiable with each customer.

Contact Information

Objective Systems SF AB   + 46 8 730 45 30
Torshamnsgatan 39
Sweden   [provider]

(Note: ObjectOry is a trademark of Objective Systems).

3.23.7 REFERENCES

[Jaco87] I. Jacobson, "Object Oriented Development in an Industrial Environment", Proceedings of OOPSLA 1987 Conference.

[Jaco86] I. Jacobson, "Language Support for Changeable Large Real Time Systems", OOPSLA86, ACM, special issue of SIGPLAN Notices, Vol. 21, No. 11, Nov. 1986.

3.24 OOA -- Object Oriented Analysis

3.24.1 BACKGROUND

Synopsis

Object Oriented Analysis represents the merging of an object-oriented approach and a data structure approach with entity-relationship modeling. It provides a systematic method for problem analysis of real-time, scientific and business-oriented systems based upon a rigorous, recorded development process. Changes are propagated through all relevant steps and reviewed at each step. "Information", "state" and "process" models are used as part of the method to represent aspects of the target system.

History

Developed by Sally Shlaer and Stephen J. Mellor, the method was first used with respect to a deliverable system in 1982. It is based upon entity- relationship modeling and information modeling.

There are other object-oriented analysis methods by the same name, OOA. In particular, see the description of the OOA method by Smith and Tockey (designated OOA/ST in the catalog).

3.24.2 DESCRIPTION

Object-Oriented Analysis (OOA) primarily addresses requirements definition and clarification and system specification, as well as providing guidelines for system design, risk/cost assessment, test planning, and project planning and tracking. OOA utilizes a data-oriented approach that does not use functional decomposition or stepwise refinement. It uses concepts from object-oriented and structured approaches, including information hiding, abstract data-types, inheritance, and module coupling and cohesion. Intended to provide a simple notation, it does not allow continuous processes.

There are four important steps which make up the OOA method. The first is termed Information Models. In this step, the conceptual entities of the problem are identified and formalized as objects and attributes. Significant emphasis is placed upon formalizing the relationships between objects. A model is developed and is depicted graphically. Textual descriptions are used to define the model's semantics.

The second step, State Models, formalizes the life or event histories of objects and relationships. The developer describes state models as being able "... to communicate by means of events; this communication is made orderly through a layering concept. State diagrams and transition tables are employed in this step."

Step three, Process Models, makes use of data-flow diagrams to develop the processes required to drive the objects through their event chains.

The final step, Boundary Statement/Requirements Definition, determines what information and processes will be within the automated system as opposed to those which will be carried out by operators or other external agents.

3.24.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

The method is well-suited for applications in areas such as process control, real-time, scientific, systems programming, distributed processing, data processing, and large scale simulation. It has been used for small, medium, and large projects, including election management software, an aluminum rolling mill, credit card billing, laser isotope separation, fiber products manufacture, and war games. It has been used in developing between 21 and 100 delivered systems by an estimated 51 to 100 organizations. The most frequently used coding languages are Fortran/Ratfor, C, Ada, and Pascal.

Target Constraints

OOA addresses timing constraints and fault-tolerance issues as part of the analysis. It uses finite state machines to model all concurrency, but does not allow continuous processes.

Modes of Expression (Tables 9,10)

OOA requires the use of finite-state diagrams, data-flow diagrams, and entity-relationship diagrams, and encourages the use of specified documentation templates and mathematical notation. It provides mapping rules for deriving the state model from the information model and data-flow diagrams from the state model.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

The method requires incremental or evolutionary development to clarify system requirements and may be used in conjunction with rapid prototyping. It requires data-structure, data-flow, and control-flow analysis.

Other Technical Aspects

OOA addresses changeability with its provisions for a rigorous, recorded stepwise development process and its requirement that changes be propagated through all relevant steps and be reviewed at each step. The method requires analysis of the problem domain and partitions the problem so that fewer program units are affected by requirements changes. The developer states that the analysis performed is, therefore, relatively independent of changes to specific requirements and that use of the method assists one in anticipating and minimizing the effects of requirements changes. The requirements for the use of a data dictionary and specific directions provided for maintaining a traceable record of technical decision-making also assist in assuring consistency and completeness of changes.

3.24.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

The method provides guidelines for analyzing risk and assessing complexity, as well as for project planning and tracking. The project plan is represented in the form of a project matrix.

Communication Channels

OOA is designed to simplify the communication of specifications and of the constraints imposed by the requirements to management, the client, and within the development team. Its emphasis on information modeling is intended to facilitate communication within the development team and between the client and the development organization. Communication between management and the technical development team is facilitated through the use of a project matrix.

The client is involved early in the analysis phase. This involvement may range from full participation in the phase, including information modeling, to review of models prepared by the developers. Incremental development is required, and thus allows for review and feedback on the evolving system from the client, management, and others involved with the development effort.

Quality Assurance (Tables 12,14,15)

OOA prescribes specific directions and procedures for software quality assurance. Its rigorous requirements for development and changes, as well as the use a data dictionary, ensure continuing consistency throughout the development process. Although it does not require a test plan, OOA does provide guidelines for test planning and requires generation of tests based on system requirements.

The developer stated that the use of the information model and state models assists in the early detection of errors. The information model reveals symantic inconsistencies. State models are easily analysed for errors.

Encouraging design reviews, code walk-throughs, and change control board review, OOA provides specific directions for maintaining records of specification/design options considered, any trade-off studies, and the rationales for decisions and requires maintaining records of all changes related to specification/design decisions. It requires some form of configuration management for the work products of the development process.

Documentation Formats (Table 16)

OOA requires the production of the following documents and allows tailoring of their formats: requirements definition, functional specification, behavioral specification, architectural specification, and interface specification. The requirements definition is produced from user responses to computer-directed prompts, while the functional and behavioral specifications are automatically generated based on data produced from other steps in the method. The use of a tailorable data dictionary is also required.

3.24.5 EASE OF USE

Technology Insertion

The minimum qualifications needed by a development team leader would include a bachelor's degree with three to five years of development experience, a working knowledge of two programming languages, and experience with two software systems. An understanding of relational theory, entity relationship/information models, finite state machines, and data flow diagrams is required. These are covered within the method training.

The developer estimates that a project manager would need about five days to acquire an understanding of the major features and benefits of the method. Developers with five or more years' experience would need about ten days to learn the essentials of the method. Experienced system developers would take about three to six months to reach expert-user level.

Assistance in learning to use the method is available in the form of classroom tutorials, on-site consulting by the vendor and by independent consultants, a users' support group, periodic technical updat^es, and related publications from third-parties.

Automated Facilities

The method provides a framework for customizing the software process with the use of other methods and tools. Available tools include Teamwork from Cadre Technologies and HP/Teamwork from Hewlett-Packard. According to the developer, these products support all the required activities of this method, and in particular, automated support is available for finite-state diagrams, data-flow diagrams, entity-relationship charts, hierarchy charts, the data dictionary, and tailorable documentation templates. Documentation support, as well as project management support, is provided through interfaces to other tools.

3.24.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

The hardware/software configuration required would be dependent upon the automated tools selected.

Acquisition Costs

The cost to acquire the method and required components is $2,500 for a single user, and $1,250 per person for low- and high-volume users. Cost for technical training is the same. A management overview costs $5,500 for 25 people.

Contact Information

Project Technology, Inc.	415-845-1484
2560 Ninth Street Suite 214	
Berkeley, CA 94710	[Developer]

3.24.7 REFERENCES

[Shla88a] S. Shlaer and S. Mellor, Object-Oriented Systems Analysis: Modeling the World in Data. Englewood Cliffs, NJ: Prentice-Hall, 1988.

[Shla88b] S. Shlaer, S. Mellor, D. Ohlsen, and W. Hywari, "The Object-Oriented Method for Analysis", Proceedings of the Tenth Structured Development Forum (SDF-X), San Francisco, CA, Aug. 1988.

3.25 OOA/ST -- Object Oriented Analysis

3.25.1 BACKGROUND

Synopsis

This method addresses the activities associated with requirements definition or clarification and system specification. It calls for specifying the requirements of the system under development in terms of essential object classes. Requirements thus stated are intended as input to an object-oriented approach to design; by maintaining the same object-orientation, the need to shift perspective between analysis and design is eliminated.

History

This method grew out of a research project at Boeing Computer Services. The project included investigating methods' suitability for software requirements analysis and software design. The method was first used for a deliverable system in 1988.

Boeing's OOA is an extension of information modeling, data-flow modeling, and finite-state modeling. The principal architects of this method are M. K. Smith and S. R. Tockey. In this catalog, their method is called OOA/ST to distinguish it from the OOA by Shlaer and Mellor. See also [Shla87].

3.25.2 DESCRIPTION

Central to object oriented analysis is the notion of an object. In Object Oriented Analysis, OOA/ST, an object may be recognized by the data it carries, its behavior, and the processing it performs. However, of these three aspects, only the aspects which are important in the problem domain are modeled in the system under development. Additionally, there is another aspect, the role that the object performs, that helps to characterize an object. This role may be expressed in terms of the operations which an object can perform on request, or performs in response to an event or condition.

There are three essential activities which comprise OOA/ST: 1) Object identification and specification, 2) Object communication, and 3) Object classes and operations. In the first of these, objects in the problem domain are discovered and described in terms of the information, behavior, and process models described above. Specifying object communication entails acquiring an understanding of the communication interfaces which may exist between object classes, providing a description in terms of messages and events passing between the classes. The third activity, identification of class operations, bridges the gap between analysis and design phases by helping to reconstruct information gained from the first two activities into a form which identifies the operations on each object class.

3.25.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

The developer reported that this method is well-suited for or compatible with a number of application areas; however, it was reported that OOA/ST would be inappropriate for applications involving image processing or pattern recognition.

There are two projects for which the method is being used within Boeing Computer Services: one of these parallels another effort to develop a production system through functional decomposition. The intent of parallel efforts is to produce requirements and design specificaitons that can be compared. The other project is a redevelopment of a large FORTRAN simulation program which models aircraft traffic flow through an airport.

The method is intended for projects of all sizes; it has been used for small and medium-sized applications. Ada is the implementation language planned for use on the aircraft traffic simulation project.

Target Constraints

OOA/ST provides specific steps for addressing concurrency and fault- tolerance issues. In terms of concurrency, each instance of each object class is allowed to be at any point in its behavioral lifecycle. The existence of, and response to, fault-related events may be included in the specification.

Modes of Expression (Tables 9,10)

The method strongly encourages iconographical representation in terms of a number of diagrams: finite-state, data-flow, entity-relationship, object communication and object operation diagrams. The developer considers flowcharts, HIPO charts, and Nassi-Shneiderman charts to be inconsistent with the method.

The method provides rules for consistency between the representation models it uses, namely between the Information model (entity-relationship), the behavior models (finite-state machines), the process models (data-flow diagrams), communication model (Oject interaction/communication diagrams), and the object operation model (modified Booch diagrams). Guidelines exist in the method for mapping from analysis to design, i.e., mapping entity-relationship diagrams into Booch and Buhr diagrams.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

Incremental or evolutionary development is strongly encouraged in order to clarify system requirements or behavior. Required for analysis are data- structure analysis, data-flow analysis, and control-flow analysis.

Other Technical Aspects

By using object-class partitioning of requirements, the method is seen to assist in reducing the effort needed to incorporate changes in the requirements. This is because the requirements are stated exactly once or isolated, and because requirements are partitioned around other closely related requirements.

3.25.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

This method does not address activities associated with project management.

Communication Channels

In the opinion of the developer, the features of OOA/ST designed to facilitate coordination and communication between all parties concerned with a project are the fact that the method is highly graphical, precise, unambiguous, and uses object partitioning. In addition, the several models represent the information from several perspectives and at various levels of detail. Provisions for involving the client in the software process are interviews for gathering requirements and OOA/ST document walkthroughs.

Quality Assurance (Tables 12,14,15)

Not applicable.

Documentation Formats (Table 16)

The method requires that documents be produced for requirements definition, functional specification, behavioral specification, and data dictionary. For all these documents, the format is tailorable within the method.

3.25.5 EASE OF USE

Technology Insertion

The major theoretical constructs which should be understood by an experienced developer in order to successfully use the method were given as entity-relationship-attribute modeling, data-flow diagrams, finite-state machines, and objects/object classes, involving information hiding and abstract data types.

Training is available as overview presentations, on-site consulting by independent consultants (for related methods), and related publications from third-parties. The developer estimated that a project manager could acquire an understanding of the major features and benefits of the method in one day; an experienced developer would require five days in order to learn to use the essentials and six months to become an expert user.

Automated Facilities

The method is not directly supported, but some integrated toolsets can be used. The developer listed Cadre Technologies as having such tools, namely Teamwork/IM, /SA, and /RT, which support information, process, and behavior modeling.

3.25.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

Not applicable.

Acquisition Costs

Not applicable.

Contact Information

Boeing Computer Services 206-237-4236
P. O. Box 3707, MS 77-87
Seattle, Washington 98124 [Provider]

3.25.7 REFERENCES

[Smit88] M. K. Smith and S. R. Tockey, "An Integrated Approach to Software Requirements Definition Using Objects", Proceedings of the Tenth Structured Development Forum, San Francisco, Ca., August 8-11, 1988; also listed in Proceedings of Ada Expo '88, Anaheim, Ca., Oct. 9-12, 1988.

3.26 OOD -- Object Oriented Design

3.26.1 BACKGROUND

Synopsis

Object-Oriented Design is a method that focuses upon the design and implementation aspects of the software process. In the process of decomposing a system, this method relies upon the concepts of classes and objects as key units of abstraction. Objects are taken to be entities whose behavior is characterized according to the actions they require or undergo. A class serves to factor the common properties of a set of objects and specify the behavior of all instances. In object-oriented approaches, objects are modeled in software, which is analogous to developing a computer simulation. It is necessary to couple the method with some form of requirements analysis [Booc86].

History

Grady Booch is the developer of this method, which was first used with respect to a deliverable system in 1982. Object-oriented approaches have roots tracing back to the SIMULA programming language and research efforts in the late 1960's and early 1970's by Alan Kay, which led to the Smalltalk language.

3.26.2 DESCRIPTION

Object-Oriented Design (OOD) addresses such activities as preliminary design, simulation, prototyping, detailed design, coding, testing, and maintenance. To do this, it devises a model of a system based upon real entities.

The development process includes the following steps:

- Identify the objects and their attributes;
- Identify the operations that may be meaningfully performed on the object or by the object;
- Establish visibility of each object in relation to other objects;
- Establish the interface of each object;
- Implement each object.

The decomposition of systems into components is considered from a different perspective than that of non-object-oriented methods. According to MacLennan [MacL86]: "Whereas functional programming concentrates on timeless mathematical relationships, object-oriented programming addresses directly the behavior of objects in time." Objects are entities which assume different states, are characterized by their actions in relation to other objects, are instances of some class, are denoted by names, have restricted visibility with respect to other objects and may be viewed either by their specifications or their implementations. Objects are classified as actors, agents or servers according to the relationship they have with other objects.

OOD builds upon the concepts of abstract data types. However, programming with abstract data types tends to deal with passive objects and the operations suffered by these objects, while OOD adds actor objects that act without stimulus from other objects and is also concerned with the operations that an object requires of other objects. The purpose of this view, as described in [Booc86], is "to decouple the dependencies of objects, especially when coupled with a language mechanism such as Ada generic units." OOD is also based upon the use of information hiding, genericity, inheritance and module coupling/cohesion, although in Booch's view, development without inheritance still constitutes object-oriented development.

The notion of an object parallels the notion of a component of a system. The development of software through OOD parallels the development of a software simulation. If one were simulating a system in software, for example, then the designer might build the software modules so that they emulate the behavior of the actual system components. The complete simulation is then built by assembling the modules into one software system where the module interactions are designed to emulate component interactions occurring in the actual system. It has been said that object-oriented programming is the systematic treatment of programming as simulation [MacL86]. Thus, the above sketch also outlines the activities which would take place with an object-oriented approach.

3.26.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

OOD is intended for use in developing real-time embedded systems, distributed processing, systems programming, data processing, and large scale simulation. It has been used in an estimated 51 to 100 organizations for developing between 21 and 100 delivered systems, including systems for ship- board command and control, banking systems, radar systems, geophysical research tools, a software development environment, and reusable software components. OOD has been used most frequently in conjunction with object- based and object-oriented programming languages, such as Ada, Smalltalk, C++, Object PASCAL, and the Common LISP Object System.

Target Constraints

OOD prescribes steps for handling requirements of the target system. *Timing constraints* are handled through the use of annotated state machines and Petri nets, as well as the use of software components available from the developer. Hardware diagrams are used to capture special features of the target hardware architecture. Special features of the target operating system may be represented with classes specifying the interfaces during the design process. Class and object diagrams capture concurrency semantics. Exception semantics are expressed for each abstraction to handle fault-tolerance issues. Spatial constraints are affected by the choices among software components. Security of access is addressed through the use of visibility restrictions. OOD assists in porting end-product systems to different target configurations through the distinct division made between the logical and physical design of a system.

Modes of Expression (Tables 9,10)

OOD requires the use of specified documentation templates, narrative overviews of modules, and PDL. It is based on the use of finite-state diagrams, data flow diagrams, Petri nets, and an object-oriented diagramatical notation, as developed by either G. Booch or R. Buhr.

OOD specifically provides steps for the evolution of a design and prescribes mapping rules for translating from data flow diagrams to class diagrams and object diagrams, as well as from class diagrams to object diagrams to architecture diagrams to hardware diagrams.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

OOD requires incremental or evolutionary development. It encourages rapid prototyping, dynamic animation, simulation, and the use of executable specifications to clarify system requirements or behavior.

Other Technical Aspects

OOD addresses changeability, reverse engineering, and reusability as follows. OOD is intended to reduce the effort needed to fully incorporate changes in the requirements. Primitive classes and objects are generated that map to a small set of requirements; changing requirements generally involves inventing new objects or altering the behavior or existing ones. The traceability between the logical and physical design is claimed to make reverse engineering possible. That is, it is possible to recreate the design from the implementation. The developer states in [Booc86] that "there is a basic relationship between reusable software components and object-oriented development: Reusable software components tend to be objects or classes of objects." The identification of patterns of classes is an integral part of the OOD process. The developer has established a library of reusable objects.

3.26.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

The method provides guidelines for analyzing risk, assessing complexity, tracking project progress, and configuration management. It requires estimating initial cost, projecting cost of completion, providing incremental data about expenditures, as well as project planning and reliability estimation.

Communication Channels

The precise specification of abstraction interfaces and abstraction relationships is designed to facilitate and coordinate communication within the development team. The development of executable prototypes is intended to facilitate and coordinate communication between management and the technical development team, as well as between the client and the development organization. The executable prototypes are used as milestones in the development process.

Quality Assurance (Tables 12,14,15)

OOD provides specific directions and procedures for checking unit interfaces. The unit interfaces are generated early in the design process in order to assist in the detection of inconsistencies and/or errors. OOD requires or provides guidelines for some specific testing activities.

Documentation Formats (Table 16)

The method specifies a fixed format for the documentation of behavioral, architectural, and interface specifications, as well as system structure charts. Design documentation and internal program documentation formats are tailorable.

3.26.5 EASE OF USE

Technology Insertion

Successful use of the method by an experienced developer requires an understanding of abstract data types, information hiding, finite state machines and object-oriented programming. The minimum qualifications needed by a development team leader for use of OOD include a bachelor's degree, with an advanced degree preferred. Additional qualifications include three to five years of development experience, a working knowledge of at least two programming languages, and development, programming, or maintenance experience with three to four different software systems. Although more experience is preferable, the method's developer felt that a user with too much experience may be less likely to make the conceptual transition required to use the method.

Training is available in the form of demonstrations and presentations, on-site consulting by the developer and independent consultants, video tapes, and through reference to various books and technical articles. The developer estimates that a project manager can acquire an understanding of the major features and benefits of the method in three days. It would take an experienced developer one week to learn to use the essentials of the method. To achieve the level of expert, an experienced developer would take six months.

Automated Facilities

Automated support tools are currently under development.

3.26.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

No specific requirements were listed by the developer.

Acquisition Costs

The method is effectively in the public domain.

Contact Information

Grady Booch 303-986-2405
c/o Rational
835 S. Moore Street
Lakewood, Colorado 80226 [Developer]

3.26.7 REFERENCES

[Booc87] G. Booch, Software Components with Ada. Menlo Park, CA: Benjamin Cummings Publishing Co., 1987.

[Booc86] G. Booch, "Object-Oriented Development", IEEE Transactions on Software Development, Vol. SE-12, No. 2, Feb. 1986, pp. 211-221.

[Booc82] G. Booch, Software Engineering with Ada. Menlo Park, CA: Benjamin Cummings Publishing Co., 1982.

[MacL86] B. MacLennan, Principles of Programming Languages, 2nd Ed. New York, NY: Holt, Reinhart and Winston, 1986.

3.27 PAISLey -- Process-oriented, Applicative, Interpretable Specification Language

3.27.1 BACKGROUND

Synopsis

The PAISLey system is an executable language for describing real-time and embedded systems as a set of concurrent processes. Execution of a PAISLey description is meant to simulate the behavior of the described system rather than to implement it. PAISLey is supported by a set of software tools, forming an environment for analyzing and executing specifications.

History

Work on PAISLey was begun by Pamela Zave at the University of Maryland, with some early contributions to the concepts by D.R. Fitzwater. Since 1981, it has been expanded at Bell Laboratories. William Schell contributed in the effort of creating the PAISLey language execution system and associated tools. The method was first used with respect to a deliverable system in 1985.

PAISLey was formed by merging two distinct models of digital computation: asynchronous processes and functional programming. Asynchronous processes were first described by Horning and Randell [Horn73] as an abstraction of concurrency in multi-programming systems; they are used to represent the developing system.

3.27.2 DESCRIPTION

PAISLey represents a Process-oriented, Applicative, and Interpretable Specification Language (PAISLey). PAISLey is an operational specification language, i.e., it specifies a system in terms of an "abstract program" that produces the same behavior as the specified system. PAISLey allows the user to create an explicit model of a proposed system that interacts with an explicit model of the system's environment. Both submodels consist of sets of asynchronously interacting digital processes. The entire model is executable and the internal computations of the processes are specified with a functional language.

"Applicative" (or "functional") languages are defined in [Zave82] as "those based on side-effect-free evaluation of expressions formed from constants, formal parameters, functions, and functional operators." PAISLey takes advantage of both applicative and process-oriented constructs.

PAISLey is intended to support the operational approach to software development; that is, one where behavior is simulated. The resultant life-cycle model views the activities of software development in terms of 1) problem understanding, 2) creation of an operational specification, 3) transformation of the specification, and 4) realization of the solution system [Zave84]. This approach to specification is described, particularly for real-time systems, in [Zave82].

The features of the language are described in [Zave86]. The language is designed for coping with complexity and change in the domain of embedded systems, where performance and resource requirements are critical. It addresses timing and concurrency issues. The developer describes PAISLey as having the following features: 1)both synchronous and asynchronous parallelism free of mutual-exclusion problems, 2)encapsulated computations, 3)ability to execute incomplete specifications, 4)executable timing constraints, 5)bounded resource consumption, 6)automated consistency checking, and 7)coherence and simplicity of notation. Support for modularity, especially abstract data types, is one of the features the developer suggests should be considered for enhancement of PAISLey.

PAISLey primarily addresses the activities of problem definition, requirements definition, specification, and preliminary design. It is intended to support simulation, prototyping, consistency checking, requirements definition, and testing. PAISLey's provisions for rapid prototyping may be used to clarify the behavior of the system with the software client and encourage incremental addition of new features. With PAISLey's provisions for execution of incomplete specifications, the specification or a fragment can be demonstrated to the client.

3.27.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

The developer states that the method is well-suited for use in the areas of embedded systems/process control, time critical/real-time applications, and systems programming. It has been used on less than five small and medium projects developed in the C language under UNIX, including a real-time transmission controller.

The use of PAISLey to model a proposed system is independent of the ultimate implementation language. The specification in the PAISLey language may be viewed as an executable design or abstract program to be translated manually or automatically into the implementation language. Alternately, the PAISLey specification could be used merely as an adjunct to a traditional requirements specification document, with design and coding done independently of PAISLey. Automated transformations and code generation are not currently available. Manual translation was done for an experimental project.

Target Constraints

The method is intended to incorporate timing constraints, operating system constraints, and concurrency issues. A timing constraint can be attached to any mapping in a PAISLey specification. The evaluation time of the mapping is viewed as a random variable, for which an upper bound, lower bound, and/or distribution may be specified. All timing constraints are interpreted in simulated time, so that execution of a specification is automatically a performance simulation as well.

Modes of Expression (Tables 9,10)

The method requires and provides automated support for the use of the PAISLey formal specification language and mathematical notation. The specification is executed to provide a simulation of the operational behavior of the system being modeled in the form of textual output.

The PAISLey language is an executable specification language for describing digital systems as a set of concurrent processes, where each process has a state and goes through a continuing sequence of discrete state changes. Processes in PAISLey specifications can be representations of objects such as system functions, data, buffers, and interactive interfaces. PAISLey is compatible with the use of finite-state diagrams, Petri nets, data-flow diagrams, control-flow diagrams, entity-relationship diagrams, or flow charts.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

The method is based upon the use of rapid prototyping, simulation, and executable specifications, and encourages the use of dynamic animation, incremental development, data-structure analysis, data-flow analysis, and control-flow analysis.

Other Technical Aspects

Not applicable.

3.27.4 PROJECT CONTROL AND COMMUNICATION

Project Management

PAISLey does not address project management.

Communication Channels

Trained analysts can communicate with people having non-technical backgrounds using simplified diagrams and narrow views derived informally from the current PAISLey specification. The developer judges the current PAISLey specification to be difficult for communicating the design outside the development team.

Quality Assurance (Tables 12,14,15)

PAISLey provides support for assessing conformity of the developing software to system specifications by:

- providing guidelines for test planning at one or more precise points in the software process;
- providing guidelines for generation of tests based on system requirements;
- providing guidelines for prescriptive checking of interfaces;
- incorporating an executable model;
- encouraging data structure, data-flow and control-flow analysis of dependencies;
- using a formal specification language.

The method makes the relationships among data, functionality, performance, and resources explicit. Validation can incorporate: the use of an informal summary derived from the formal specification, the execution of the interpreter, or the generation of a prototype. Validation can utilize the specification or an incomplete fragment. The interpreter can also be used as a performance simulator.

Documentation Formats (Table 16)

PAISLey does not address external documentation. It prescribes a fixed format for requirements definition, functional specification, behavioral specification, and architectural specification.

3.27.5 EASE OF USE

Technology Insertion

Training is available in the form of hands-on demonstrations, overview presentations, classroom tutorials, user manuals, "hot-line" service, and related publications from third-parties. The developer estimated that it would take three days for a project manager to learn the basics of the method, five days for an experienced developer, and that it would take three months for an experienced developer to achieve the level of expert user.

The minimum qualifications needed by a development team leader include a Bachelor's degree, three to five years of development experience, a working knowledge of two programming languages, and experience with two different software systems. Successful use of the method requires an understanding of concurrent processes and functional programming.

Automated Facilities

PAISLey cannot be separated from the associated tools without losing the essence of the method. The PAISLey environment provides semantic and/or syntactic analysis for the formal specification language. The interpreter provides for simulation or execution. The consistency checker provides data structure analysis. The parser and cross referencer provide control flow analysis. It utilizes Unix facilities for generating, editing, and storing specifications.

3.27.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

PAISLey software tools are embedded in the Unix operating system and follow the Unix style for program interfaces. The method utilizes Unix facilities for generating, editing, and storing specifications. PAISLey runs on System V and 4.2 BSD versions of the Unix system, and has been executed on Sun, DEC/VAX, ATT/3B, Amdahl 470, and M68000 processors.

Acquisition Costs

The unit cost to acquire the method and required components is $2,800. Unit cost for technical training is $3,000, and unit cost for a management overview is $1,500.

One source license covers all usages at a site.

Contact Information

Pamela Zave (201) 582-3080
AT_&T Bell Laboratories
Computer Technology Research Labs
600 Mountain Avenue, Room 3D-426
Murray Hill, NJ 07974 [developer]

3.27.7 REFERENCES

[Horn73] J. J. Horning and B. Randell,"Process Structuring", ACM Computing Surveys, Vol. 5, No. 1, March 1973, pp. 5-30.

[Zave86] Pamela Zave and William Schell, "Salient Features of an Executable Specification Language and Its Environment", IEEE Transactions on Software Engineering, Vol. 12, No. 2, Feb. 1986, pp. 312-325.

[Zave84] Pamela Zave, "The Operational Versus the Conventional Approach to Software Development", Communications of the ACM, Vol. 27, No. 2, Feb. 1984, pp. 104-118.

[Zave82] Pamela Zave, "An Operational Approach to Requirements Specification for Embedded Systems", IEEE Transactions on Software Engineering, Vol. 8, No. 3, May 1982, pp. 250-269.

3.28 PAMELA 2 -- Parts Assembly Method for Embedded Large Applications

3.28.1 BACKGROUND

Synopsis

This method covers the software requirements phase of development as well as the design and implementation phases. It provides a graphical notation (called Ada graphs) based on Ada's semantics for communicating program specifications and design, and incorporates dataflow, process-oriented, and object-oriented approaches.

History

PAMELA 2 is a second-generation method based on PAMELA. Both methods were developed by George W. Cherry. PAMELA 2 assimilates several other approaches and methods, including PAISley, Statecharts, Structured Analysis/Real Time, Structured Design/Real Time, Object-Oriented Development, and MASCOT 3. The method was first used in 1988 for the development of a deliverable system.

3.28.2 DESCRIPTION

PAMELA 2, Parts Assembly Method for Embedded Large Applications, has three basic views of the software under development: 1) a users' view, shown with a specification graph; 2) a bottom-up implementation view, conveyed with a library graph; and 3) a behavioral view, shown with processing graphs. The method provides guidelines for developing these views using PAMELA 2's iconographical representations and text where appropriate.

The specification graph shows the requirements for the part of the software represented in the graph, as well as the interfaces which result from connecting that part to other parts of the system. In composing the library graph, the designer attempts to identify library objects which encapsulate information about the problem to be solved; the emphasis in this step is to identify reusable components.

The processing graphs are used iteratively to refine the behavior of components and subcomponents. PAMELA 2 provides icons for representing all the semantics of the Ada language, including control icons and other icons to show either sequential or concurrent processing. These icons can express the semantics of synchronization between processes and interprocess communication by means of parameterized (data flow) calls. There are icons for conditional, priority, and timed calls, icons for conditional and guarded waits, and an icons for exceptions. Other icons allow representation of data stores or shared storage. The icons also distinguish between generic packages, super library units, and hardware objects.

3.28.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

The developer rated the method well-suited for all the applications shown in Table 5. The method is intended for use on projects of all sizes, and has been used for small and medium-sized projects. Ada is the implementation language most frequently used when coding systems developed with PAMELA 2, which has been used to develop approximately five delivered systems, within between five and twenty organizations.

Target Constraints

The developer stated that the method prescribes steps for handling several types of requirements of the target system, including timing and spatial constraints, special features of the target hardware architecture and operating system, concurrency and fault-tolerance issues, and security of access. By separating hardware-dependent parts from application-dependent parts, the method assists with portability.

Modes of Expression (Tables 9,10)

PAMELA 2 requires a program design language and a formal specification language. It also requires finite-state diagrams, petri nets, data-flow diagrams, control-flow diagrams, flowcharts, hierarchy charts, and Buhr and Booch diagrams. Its own iconographical representations constitute a major form of expression as well.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

A number of techniques are strongly encouraged for clarifying system requirements: rapid prototyping, dynamic animation, simulation, incremental or evolutionary development, and executable specifications. Code walk-throughs are a required technique, and several other analysis and review techniques are strongly encouraged.

Other Technical Aspects

By having the same process used for development as for re-development, the method is seen as assisting the incorporation ot changes within the requirements. The method has an explicit reuse/reusability first step in the development phase of each part.

3.28.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

PAMELA 2 prescribes specific procedures for analyzing risk, estimating initial cost, projecting cost of completion, providing incremental data about expenditures, and tracking project progress, as well as for project planning, scheduling and/or manpower loading, allocation of personnel to tasks, allocation of development resources, configuration management, and reliability estimation.

Communication Channels

The developer regarded the standard graphical representations of the method as facilitating communication within the development team and between the client and the development organization. The client is involved during the software development process at reviews of the SRS, architectural design, and detailed design, all of which are based on Ada graphs.

Quality Assurance (Tables 12,14,15)

The method provides a framework for accomplishing several types of testing activities: test planning, generation of tests based on system requirements, unit/integration testing, field or acceptance testing, generation of test data, and prescriptive checking of interfaces. The fact that all graphs are consistent by construction was cited as leading to early detection of inconsistencies and/or errors. The method also provides specific directions for recording information relative to technical decision-making during the software development process.

Documentation Formats (Table 16)

The Ada graphs incorporated in the method constitute a form of documentation for the system under construction. Other documents are required to be produced, whose formats are tailorable within the method.

3.28.5 EASE OF USE

Technology Insertion

The developer estimated that a development team leader would need, at minimum, the following for successful use of the method: a bachelor's degree, three to five years of development experience, knowledge of two programming languages (Ada being one of them), and experience with one software system. Theoretical constructs which should be understood by an experienced developer in order to successfully use the method are the semantics of Ada, finite state machines, task idioms, communicating processes, and the object-oriented paradigm.

Training assistance is available in the form of hands-on demonstrations, overview presentations, classroom tutorials, on-site consulting by the vendor or independent consultants, a hot-line service, user manuals, a users' support group, and periodic technical updates. The developer estimated that one or two days would be required for a project manager to acquire an understanding of the major features and benefits of the method, and one week for an experienced developer to learn to use the essentials. One month would be required for such a developer to achieve the level of expert user.

Automated Facilities

HyperPAL by Thought**Tools is under development, and is intended to support all the activities of PAMELA 2.

3.28.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

This information was not provided.

Acquisition Costs

The unit cost to acquire the method and required components is $5,000. Technical training or a management overview each costs $1,000.

The licensing policy is customized to the customer.

Contact Information

George W. Cherry 703-437-4450
Thought**Tools, Inc.
P. O. Box 2429
Reston, VA 22090 [Developer]

3.28.7 REFERENCES

"Reference Manual for the PAMELA 2 Specification and Design Language and Method" is available from Thought**Tools.

[Authors' note: due to publishing deadlines, PAMELA 2 does not appear in the tables in Chapter 6]

3.29 **PDL/81** -- Program Design Language/81

3.29.1 BACKGROUND

Synopsis

PDL/81 is a method which uses a structured-English program design language to assist in the creation of both the preliminary and detailed design for a software system. The method emphasizes top-down design and the use of structured programming constructs. The method provides an automated facility which facilitates the creation of design documentation.

History

PDL/81 is the latest version of a method developed at Caine, Farber, Gordon, Inc. during the early 1970's. The method was first used for a deliverable system in 1973.

3.29.2 DESCRIPTION

Program Design Language/81, PDL/81, is a software development method which is primarily concerned with the design phase. The method also addresses aspects related to documentation and maintenance. The method is founded upon a control-oriented approach and the use of functional hierarchy/decomposition, stepwise refinement, process abstraction, and structured programming. The design is developed and presented in a top-down manner with an emphasis on understandability.

The method stresses development of a complete design before code is written. The complete design should contain:

- Definitions of all external and internal interfaces;
- Definitions of all error situations;
- Definition of all global data;
- Definition of all control blocks;
- Identification of all procedures and procedure calls;
- Specification of the processing algorithms of all procedures.

PDL is the program design language which is used to document the design. PDL uses structured English, providing a formal language structure which allows the designer to communicate ideas to other people. The language allows construction of a problem solution in a top-down manner. Thus, it is possible to review the first levels of design, and easily incorporate modifications. In [Cain75] the developer states: "...the basic readability of a PDL design means that clients, management, and team members can both understand the proposed solution and gauge its degree of completeness."

An automated tool, called a processor,is provided by the method. The processor assists in producing the design in PDL. Input to the processor consists of control information and designs for procedures, called design segments. The output of the processor is a working design document. The language encourages and supports design constructs which relate directly to structured programming constructs. However, the designer can use his discretion as to what input is provided for a design segment.

3.29.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

Software development areas for which the method is well-suited include embedded systems, process control, or device control, distributed processing/networks, real-time or time-critical systems, and system programming.

The developer states that the method is best used for small to medium size projects. More than 100 organizations have used PDL/81 in developing more than 250 delivered projects of all sizes, including compilers, command and control systems, operating systems, and medical imaging systems.

The implementation languages most frequently used when coding systems developed with PDL/81 have been Fortran, Ada, C, Jovial, and Assembler. The method assumes that the implementation language provides structured programming constructs.

Target Constraints

The method does not provide specific directions for addressing particular constraints of the target system.

Modes of Expression (Tables 9,10)

PDL/81 requires the use of a program design language based upon natural language (structured English) during all phases of design. The automated processor which supports the method provides syntactic analysis for the program design language. The method does not use graphics, but rather the user is encouraged to write meaningful narrative overviews and processing descriptions. The use of a structured natural language to express the processing descriptions is intended to facilitate the evolution and communication of the design.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

Use of control-flow analysis is required by the method. The use of incremental or evolutionary development techniques are encouraged.

Other Technical Aspects

PDL/81 provides a tailorable format for internal program documentation. To aid in changeability and traceability, the developer states that the method provides an index that shows which design segments address each requirement of the system. Additionally, incorporated into each design segment is a statement of those requirements that are associated with the segment. The method assists in identifying possible reusable components due to its emphasis on meaningful natural language.

3.29.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

The method does not directly address issues involving project managemeLt or tracking. It does prescribe specific directions and procedures for assessing the complexity of the system being developed, as well as providing automated support.

Communication Channels

PDL facilitates communication among clients, management, and team members. The use of structured natural language facilitates understanding without the necessity of formal training. The client sees the design during all development stages and, since programming expertise is not required for understanding, can give timely feedback.

Quality Assurance (Tables 12,14,15)

The developer states that the ease of reading and understanding the design and the consistency checking provided by the method assist in the early detection of inconsistencies and/or errors. The use of structured natural language presents the necessary information in a suitable form for use in design reviews and code walk-throughs, as well as providing background data when necessary for Change Control Board review.

Documentation Formats (Table 16)

PDL/81 can be utilized in conjunction with customized templates to accommodate an external documentation standard. It requires, and provides a tailorable document format for the following: requirements definition, functional specification, design document, internal program documentation, and user manual.

3.29.5 EASE OF USE

Technology Insertion

Training is available in the form of on-site consulting by independent consultants, "hot line" service, and user manuals. It is estimated that a project manager can gain an understanding of the major features and benefits of the method in one day. An experienced developer would require two days to learn to use the essentials of the method and would reach the level of expert user in one month. To successfully use the method, the background of the development team leader should include two to three years college-level technical education, one to two years of development experience, a knowledge of one programming language, and experience with one different software systsm.

Automated Facilities

PDL/81 provides automated support for specified documentation templates, narrative overviews of modules, and a program design language based on structured English.

3.29.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

Appropriate configurations on which to host the method include VAX/VMS; Sun, Apollo, VAX, and most other Unix-based systems; and IBM-PC under DOS and XENIX.

Acquisition Costs

Use is licensed on a per-CPU basis. For the initial license and each multi-user machine the cost would be $6,000. The cost for an additional single-user CPU is $600, with volume discounts thereafter.

Contact Information

Caine, Farber, & Gordon, Inc. 818-449-3070
1010 E. Union St.
Pasadena, CA 91106 [Developer & provider]

3.29.7 REFERENCES

[Cain75] S. H. Caine and E. K. Gordon, "PDL - A Tool for Software Design", Proceedings of the 1975 National Computer Conference, 1975, pp. 271-276.

[Cain88] "PDL/81 - An Introduction", October, 1988. Available from Caine, Farber & Gordon, Inc.

3.30 PRIDE -- "PRIDE" Family of Products for Information Resource Management (IRM)

3.30.1 BACKGROUND

Synopsis

PRIDE offers a software development method which focuses on the design of information systems and of corporate databases. Automated tools support the development process, and also facilitate project management and documentation. Together, the set of products provides a development environment which addresses almost all aspects of software development.

History

The PRIDE set of products has been developed at M. Bryce & Associates, Inc. The first products were developed in the early 1970's, and employed structured system design techniques and the then emerging database technology. PRIDE was first used for a deliverable system in 1971.

3.30.2 DESCRIPTION

The software development process of PRIDE is called Information Systems Engineering Methodology. PRIDE-ISEM consists of several distinct phases, proceeding from feasibility to review, which make up a framework for development. The phases are:

- System Study and Evaluation: an analysis of the problem and a proposal to management for a course of action;
- System Design: a determination of the required sub-systems;
- Sub-system Design: a specification of the means of implementation;
- Administrative Procedure Changes: the preparation of a clear and detailed description of the procedures for the client;
- Computer Procedure Design: the definition of specifications for the program steps;
- Program Design: the production of efficient machine code;
- Computer Procedure Test: testing of the procedures;
- System Test: integration of the sub-systems;
- System Operation: the accommodation of possible modifications;
- System Audit: a determination of whether expectations were met.

For development of a corporate database, PRIDE uses Data Base Engineering Methodology. The method uses six phases which are centered around four models for defining the database. These models are:

- The application logical database model: a view of the enterprise objects required for a specific application;
- The enterprise logical database model: an integration of the various application logical models;
- The enterprise physical database model and the application physical database model: a specification for each of the logical models of how data is to be physically stored.

The six phases involve an initial study and evaluation, the design of the four database models, and a project audit.

The method is founded on a data structure-oriented approach, on what the developer refers to as "object-4 data model approach", and on chronological decomposition, which emphasizes the identification of timing dependencies within the data that affect the production of information from such data. The method is well-suited for use within the context of several software process paradigms; it regards systems as being built by evolution. Essential concepts to PRIDE are stepwise refinement, process abstraction, and abstract data-types.

The PRIDE family has been expanded to include EEM, Enterprise Engineering Methodology. This method implements a business planning stragegy for a corporation within five phases:

- EEM Project Planning, concerning an enterprise's mission, Business Plan, and influences outside the enterprise;
- Logical Enterprise Analysis, describing the enterprise and its environment;
- Physical Enterprise Analysis, similar to the above phase and used to perform an Organization Analysis of resources;
- Develop Enterprise Information Strategy specifying the objectives and projects that will be followed;
- EEM Evaluation, used to initiate the strategy as well as to audit the EEM project for actual time and cost versus projected.

The PRIDE products include several integrated tools which support the development process. The Project Management System (PMS) is used to support various activities associated with project management, including initiation of the project and planning. The Information Resource Manager (IRM) is used to catalog and control information resources. Among its features are a "status check" which is designed to ensure that all components are properly defined and related, and an "impact analysis" which measures the effect of a proposed change. The Automated Engineering System (ASE) provides automated assistance for design and modification of the system, expediting the administrative tasks associated with system design. The Computer Aided Planning (CAP) tool provides automated support for EEM.

3.30.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

The developer stated that the method is well-suited for applications involving embedded systems or process control, time critical or real-time processing, scientific or engineering processing, distributed processing or networks, data processing or database applications, and expert systems or artificial intelligence.

Examples of delivered systems developed using the method include inventory control, factory automation, production control, transportation systems, military systems, health & welfare applications and pension plans, telephone systems, gas pipeline distribution, customer billing, payroll and employee benefits systems. The number of delivered systems that have been built using the method is estimated at more than 250, and the number of organizations which have used the method exceeds 100.

The method is intended for use on projects of all sizes and has been used as such. The implementation languages most frequently used when coding systems developed with PRIDE are COBOL, PL/1, FORTRAN, C, Basic, Assembler, and fourth-generation database languages.

Target Constraints

The developer states that chronological decomposition addresses timing constraints and concurrency issues. The method textbook provides directions for addressing issues concerning special features of the target hardware architecture and operating system, as well as security of access issues.

Modes of Expression (Tables 9,10)

The method requires specified documentation templates and encourages narrative overviews of modules. Automated support is provided for both these textual modes. Required iconographical modes of representation are control-flow diagrams, flowcharts, and hierarchy charts. The method provides automated support for all of these representations. Use of matrix tables is also required.

Mapping rules are prescribed for translating from a system flowchart to a sub-system flowchart, and from a sub-system flowchart to a computer procedure flowchart. Transformation across phases of the software process is facilitated through the formal deliverables resulting from each phase.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

The method requires incremental or evolutionary development to clarify system requirements. Required analysis and review techniques include data-structure, data-flow, and control-flow analyses, formal proof techniques, design reviews, and Facilitated Application Specification Techniques.

Other Technical Aspects

The method assists in reducing the effort to incorporate changes in the requirements through a "change control"mechanism provided by the IRM tool. This tool keeps track of changes and assures that they have been implemented. Also, documentation can be regenerated. The method assists in ensuring consistency between specification, design or code by means of reviews and an IRM "status check" which maintains consistency.

Assistance in identification of reusable components is found in the IRM's "logical attribute" search routine for locating any component.

3.30.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

The developer reports that the method prescribes specific procedures for conducting activities involving risk assessment and cost estimation of software development, as well as involving planning, scheduling, resource allocation, skills inventory, and user billing/chargeback.

Communication Channels

The aspects of PRIDE designed to facilitate and coordinate communication among all parties are formal and informal review points, standard documentation, recommended procedures of project management, and the control point provided by IRM. The method involves the client through formal reviews at the end of each phase and through informal discussions during each phase.

Quality Assurance (Tables 12,14,15)

PRIDE provides a framework for conducting testing activities and configuration management. Early detection of inconsistencies or errors is accomplished with IRM's "status check", which notes errors and omissions.

Specific directions are provided by the method for recording and maintaining records of technical decision-making.

Documentation Formats (Table 16)

The formats and level of automated support provided by the method for its required documents vary. See Table 16 for specific information.

3.30.5 EASE OF USE

Technology Insertion

The developer states that successful use of the method by experienced developers requires an understanding of business and a capability to think about a problem. A development team leader would need less than two years of college-level technical education, no prior development experience, working knowledge of one programming language, and experience with one software system in order to use the method successfully.

Training in the method is provided in the form of hands-on demonstrations, overview presentations, classroom tutorials, on-site consulting by the vendor, on-line help facility, a "hot line" service, user manuals, and periodic technical updates. One day would be required for a project manager to understand the major features and benefits of the method, and five days for an experienced developer to learn to use the method's essentials. Six months was estimated as the time for an experienced developer to become an expert user of the method.

Automated Facilities

The vendor provides a number of tools which assist in the functions of the method. Refer to the Description section above and to Tables 9, 10, 13, and 16 for specific information.

3.30.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

Appropriate configurations for hosting the tools of the method include IBM MVS, MVS/XA, DEC VAX/VMS, and UNISYS 1100 EXEC.

Acquisition Costs

The cost to acquire the method and required components along with included technical training is $250,000. For other recommended vendor components, the cost is $10,000. A management overview is $1,500. There is a site license.

Contact Information

M. Bryce & Associates, Inc.
777 Alderman Road
Palm Harbor, FL 34683

813-786-4567

[Developer]

3.30.7 REFERENCES

The developer provides product descriptions and other literature.

3.31 PROMOD -- the Project Models

3.31.1 BACKGROUND

Synopsis

PROMOD is a software engineering environment which supports a system of selected methods through a set of integrated tools used for different phases of the software development process. Tools supporting structured analysis are used for requirements analysis and definition. Tools supporting real-time analysis provide control-oriented requirements definitions. The method of modular design is supported for the system design phase. During the program design phase, tools which support the use of a program design language are used. These tools are intended to produce a consistent and complete Logical Model in Analysis, a consistent and complete Physical Model in Design, and an Implementation Model that matches the Physical Model in the Implementation Phase.

History

PROMOD was developed at GEI - Gesellschaft fuer Elektronische Informationsverarbeitung in Germany. It was first used in the development of a deliverable system in 1980. The set of integrated tools are based on principles advocated in the following:

- Yourdon/DeMarco's method of Structured Analysis;
- Hatley/Pirbhai real-time extensions to structured analysis;
- D. L. Parnas' principles of information hiding and data abstraction;
- Parnas/Booch/Buhr methods of modular design;
- Caine, Farber, Gordon technique for pseudocode program specifications;
- N. Wirth's technique of stepwise refinement for programming.

3.31.2 DESCRIPTION

PROMOD, the Project Model, is an integrated set of automated tools which provides a software engineering environment that supports all of the technical phases of software development. PROMOD recognizes that different methods are best suited to different phases of the development cycle, and provides an environment which is intended to facilitate the transition between the use of different methods. The automated tool set of PROMOD supports and enforces the correct use of the various methods.

The PROMOD project model specifies the following phases in the software development cycle: 1) requirements analysis and definition, 2) architectural or system design, 3) program design, 4) implementation, 5) system test, and 6) operation. PROMOD has chosen to support Structured Analysis for the requirements analysis and definition phase. For the architectural or system design phase, it supports a method called Modular Design which is a combination of Structured Design and Information Hiding. Detailed design is accomplished by stepwise refinement, with several choices provided to the user.

During requirements analysis and definition (RA&D), PROMOD provides:

- A process for producing data-flow diagrams;
- A process for recording these diagrams in a data dictionary;
- A process for producing documents called minispecs;
- A process for recording data structures in detail;

- The RA&D analyzer for investigating system model interrelations and reporting on inconsistencies and incompleteness in the logical model.

The first four processes interactively provide recording and syntactical checking of the various objects, analysis of the objects, and preparation of reports in a wide variety of formats. Isolated nodes and cycles within data-flows are identified. The RA&D analyzer provides further analysis and cross-reference lists. The output of the RA&D analyzer is the Logical Model.

Using the definitions of the data-flow diagrams recorded in the data dictionary, minispecs are generated. These minispecs are a colloquial description of the various system functions, and provide an easy mode of communication with the client. Use of the system facilitates iteration and modification during the analysis phase. The real-time analysis facilities for control-oriented requirements definition enable creation of control-flow diagrams, state transition diagrams, activation tables, and a requirements dictionary.

In the architectural or system design phase, there are several processes supported which assist in construction and checking of system specifications. These processes include:

- Transformation of the system model into a modular structure;
- Definition of the specification of the modules;
- Definition of the specifications of the functions;
- Extension of the information in the data dictionary as data types;
- Checking and editing of system specifications.

Checking and editing is performed by the MD-analyzer which corrects defectively specified imports, identifies functions which are used but not defined, detects errors in the parameter lists of functions, and indicates the use of data which belongs to other functions. Additionally, the MD-analyzer edits the system specification document to provide optical configuration of modules, data, and functions, and to provide representations of module interrelations, invocation hierarchy of the functions, and cross-references for data and functions.

For the program design phase, the system provides a choice of a function specification process, a process which uses a program design language, or a process for aiding in the design of real-time systems. Which process is used will depend upon the dimension and purpose of the software project. Function and data are specified according to the method of stepwise refinement. The process investigates the constituted structures and detects errors and inconsistencies. The designer is then able to debug and elaborate program specifications.

Further tools are available to support the implementation phase, assisting in transformation of the program specifications into code, and providing automatic documentation of the code.

3.31.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

The developer rated the method well-suited for all the applications areas listed in Table 5, with the exception of a compatible rating for data processing or database and expert systems or artificial intelligence. The method has been used to develop embedded computer applications for defense, aerospace, and real-time process control, as well as for an instrumentation system and a VAX-based database and network inquiry/update system for telephone company information.

The method is intended for use on projects of all sizes and has been used for all such projects. The developer estimated the number of delivered systems developed with the method at more than 250, in more than

100 organizations. Languages most frequently used when coding systems developed with the method were FORTRAN, C, Ada, Pascal, Jovial, Assembler, and COBOL.

Target Constraints

The developer stated that textual requirements in templates are provided for handling timing and spatial constraints. Specifications templates are provided for handling special features of the target hardware architecture and operating system, concurrency and fault-tolerance issues, and security of access.

The method assists in porting end-product systems to different target configurations with generation of source code in Ada, C, and Pascal. The method supports code refinements with code-specific tools.

Modes of Expression (Tables 9,10)

The method provides automated support for a number of textual and iconographical modes of representation. The use of these techniques is dependent on the analyst and designer. In addition, the developer reported that several modes were inconsistent with the method, including Warnier/Orr diagrams, petri-nets, entity-relationship diagrams, flowcharts, HIPO charts, and Nassi-Shneiderman charts.

PROMOD prescribes detailed mapping rules for translating from one mode of expression to another. Included are rules for translating from flow diagrams to module hierarchy, from process descriptions to functions and relationships, from modules to Ada packages, or C and Pascal files, from functions to Ada procedures, and from data types to parameters. The developer mentioned that there were other translation rules as well. Across phases of the software process, the method facilitates transformation of the Logical Model to suggested design, and from the Physical Model to source code.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

PROMOD requires data-structure analysis, data-flow analysis, and control-flow analysis. It strongly encourages rapid prototyping and incremental or evolutionary development to clarify system requirements.

Other Technical Aspects

The developer reported that PROMOD assists in incorporation of changes in the requirements by graphical representation of requirements, cross-reference lists for all components, inconsistency and incompleteness checks, and global access to components and their relationships.

The method assists in identification of possible reusable components with consistent naming conventions, well-defined visible parts, and protected or hidden implementations.

3.31.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

Specific directions and procedures and prescribed for assessing complexity and for configuration management. See Table 13 for other activities supported at a more general level.

Communication Channels

The developer states that there are several aspects designed to facilitate communication within the development team and between the development team and management. These are automatic, consistent documentation produced in conjunction with rule checks, standard methods for analysis, design, and code, and interaction with other tools such as configuration control and project management.

The method facilitates communication between the development organization and the software client with well-defined requirements in the analysis phase, and transformation of portions of requirements to design and then to code for "critical path rapid prototyping". Moreover, there is automatic documentation for conducting reviews, prepared on demand, as well as an open database structure to automatically include client data.

Quality Assurance (Tables 12,14,15)

PROMOD provides a framework for test planning and generation of tests based on system requriements. It assists in the early detection of inconsistencies and errors with incremental checks on small portions of the total project, and with early visibility of the system structure. The method also provides specific directions for recording information concerning the technical decision-making during the software development process.

Documentation Formats (Table 16)

The majority of documents required to be produced by the method are tailorable in format and automatically generated based on data produced from other steps in the method. For specific documents, see Table 16.

3.31.5 EASE OF USE

Technology Insertion

The developer estimated that a development team leader, for successful use of PROMOD, would need a bachelor's degree, three to five years of development experience, knowledge of two programming languages, and experience with two different software systems. Theoretical constructs which should be understood by an experienced developer in order to successfully use the method include structured methods in general, structured analysis as a requirements tool, modular (object-oriented) design as a specification tool, and pseudocode (PDL) as a detailed design tool.

The developer has numerous means for training; see Table 18. It was estimated that a project manager would need two days to acquire an understanding of the major features of the method. An experienced developer would need five days to learn to use the method's essentials, and two months to achieve the level of expert user.

Automated Facilities

PROMOD provides automated support for the modes of representation (Tables 9 and 10), for assessing complexity (Table 13), and documentation (Table 16).

3.31.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

All VAX/VMS and PC/MS-DOS systems are appropriate for hosting PROMOD.

Acquisition Costs

There are nine products and five platforms comprising the method. Cost to a low volume user to acquire the method and required components is $6,000, and $4,000 for a high-volume user. Technical training is $1500 per day for high-volume users.

The licensing policy is per CPU.

Contact Information

ProMod Incorporated 714-855-3046
23685 Birtcher Drive
El Toro, California 92630 [Provider]

3.31.7 REFERENCES

[Hrus86] P. Hruschka, PROMOD - Motivation and Introduction. Laguna Hills, California: Promod, Inc., 1986.

3.32 PROTOB

3.32.1 BACKGROUND

Synopsis

This method consists of using high level Petri nets, or PROT nets, to provide a conceptual graphical model for discrete event dynamic systems such as manufacturing lines, production schedulers and controllers, communication protocols and real-time systems. PROTOB is a visual object-oriented programming language that is used to specify and simulate the operational behavior of the software. Automatic code generation to different target languages is a key aspect of the method. PROTOB is available for general use.

History

PROTOB was developed by Giorgio Bruno and Marco Baldassari at Politecnico di Torino. It is based on the mathematical theory of Petri nets [Petr81], [Genr81], [Reis86], and was first used with respect to a deliverable system in 1987.

3.32.2 DESCRIPTION

PROTOB prescribes specific procedures for requirements definition, system specification, design, and implementation. It is founded upon both an object-oriented and control-oriented approach. The most effective use of this method is dependent on the operational model of the software process. The developer states that the design and coding steps traditionally associated with the development process can be skipped. This is because the system allows executable tasks to be generated from the method's model according to a given configuration, each task implementing a portion of the model.

A number of concepts were listed as essential to the method, including information hiding, process abstraction and abstract data-types, structured programming, genericity and inheritance.

In particular, PROTOB is a method of specification and simulation based upon Petri net formalism. The method captures the concepts of state and of transition between states based on time and on mutual interaction. It provides both visual and textual information in a model which can be used for simulation and automatic code generation to a target language.

The components of the visual model include symbols connected by directed arcs. These symbols consist of three types: places, transitions, and sub-objects. Places, shown as circles, contain typed data structures called tokens. Text associated with each token provides name and type information of the data. Transitions, shown as rectangles, have four textual attributes associated with them:

- Predicate: the condition to be evaluated on the contents of the tokens present in the input places. Tokens satisfying this predicate enable the transition to "fire", or take place.
- Action: the operations to be performed on selected tokens during firing.
- Delay: a period of time to wait before delivering tokens to output places. The default is immediate movement of tokens from input to output places upon firing.
- Priority: a non-negative integer specifying the relative importance of the transition regarding the order of firing when more than one transition is enabled. Default priority is zero.

Sub-objects, represented as squares, provide for object-oriented decomposition of a PROT net.

Different languages to specify predicates and actions associated with transitions can be used in the textual part of the model. This textual part, written in a specific programming language, together with the PROT Net description, can be used to produce the final program for the target environment by means of a program generator.

3.32.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

The method was rated well-suited for applications involving control and concurrency, including embedded systems or process control, time critical or real-time applications, distributed processing or networks, and large scale simulation or modeling. Specific types of applications which have been developed and delivered using the method are production control and scheduling, real-time systems, communication protocols, distributed systems, and discrete event dynamic systems in general.

The estimated number of delivered systems developed with the method is between 5 and 20, distributed among the same number of organizations. The developer reported that the method is best used on projects of medium size, and has been used for medium-sized applications.

A PROT net consists of a graphical part accompanied by a textual part. This textual part depends upon the target language, which may be a procedural language like Ada, C, and Pascal, a rule-based language like OPS5, or a database language like DATATRIEVE.

There are mappings between the model's textual predicates on the one hand and actions associated with transitions and the programming language constructs chosen for the textual specification on the other hand. Ada, with its task construct, has a different mapping than Pascal. Using the PROT net graphical model, Pascal has been used to produce fast discrete event simulators. There is also a commonality between the Petri net family and rule-based languages, both of which have data-driven operations. An example of one rule-based language used with PROTOB is OPS5. Pascal has been most frequently used for coding systems developed with PROTOB.

Target Constraints

PROTOB has steps for handling both timing constraints and concurrency issues of the target system. Timing constraints can be associated with transitions in PROT nets; they can be simulated during the execution (simulation) of the model or also emulated by calling the target operating system. The model within the method can be executed via different tasks or on different processors.

Modes of Expression (Tables 9,10)

Programming languages are required as the textual modes of representation in the method, while Petri nets are required iconographical modes. The method strongly encourages the use of specified documentation templates; other forms of textual and iconographical representations are considered inconsistent with the method.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

PROTOB requires rapid prototyping, dynamic animation, simulation, and executable specifications for clarifying system behavior. It does not specifically require analysis or review techniques, although data-flow and control-flow analyses, design reviews, and code walk-throughs are strongly encouraged.

Other Technical Aspects

By running the model, changes can be immediately analyzed, thus assisting in reducing the effort to incorporate changes to the requirements. Since the system generates the code automatically, any change performed at the specification level has direct influence on the final program. The specification model is operational, thus felt to assist in the early detection of inconsistencies and errors.

Each object of the object-oriented model is a reusable building block. In this way the method assists in identification of possible reusable components.

3.32.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

PROTOB does not address issues associated with project management.

Communication Channels

The developer reported that the object-oriented decomposition of the model was designed to facilitate communication within the development team. The fact that the specification model can be executed and animated, giving the client a chance to see the behavior of the model, is designed to enhance communication between the client and the development organization.

Quality Assurance (Tables 12,14,15)

PROTOB does not provide directions for accomplishing these activities.

Documentation Formats (Table 16)

Both functional and behavioral specifications are required to be produced: the method prescribes a fixed document format for each, and nrovides automated support based on user responses to computer-directed prompts.

3.32.5 EASE OF USE

Technology Insertion

Minimum qualifications given by the developer for successful use of the method as a team leader include a Bachelor's degree, knowledge of 1 programming language, and software system, but no prior development experience. Major theoretical concepts recommended for successfully using the method as an experienced developer were Petri nets, object-oriented programming, and, optionally, compiling techniques.

Presently available training assistance includes hands-on demonstrations and overview presentations, classroom tutorials, on-site consulting by the vendor, on-line help facilities, and user manuals.

A pro: , ıanager could learn the major features and benefits of the method in one day. An experienced deveioper would need seven days to learn the essentials, while expert user level could be attained in one month.

Automated Facilities

Associated with the method are the following tools: an editor for PROT net objects, used for generating the gr. phical representation of the model in the form of extended Petri nets and for performing consistency checks; a PROT net simulator, which uses discrete event simulation techniques to execute the model, providing animation at both the PROT net object level and at the application level; and a report generator, which extracts documentation for the PROT net models and the results of simulation runs.

3.32.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

A DEC VAX with VT241 terminals, or a DEC VAXstation would be appropriate for hosting the method's automated tools.

Acquisition Costs

Unit cost to acquire the method and required components for a single user is $15,000; for a high-volume user, $10,000. Technical training costs $4,000, and a management overview is $2,000.

Contact Information

Giorgio Bruno
Associate Professor
Dipartimento di Automatica e Informatica
Politecnico di Torino
Corso Duca degli Abruzzi 24
10129 Torino, Italy

(011) 556-7003
Telefax 39-11-5567099

[Developer]

3.32.7 REFERENCES

[Brun86a] G. Bruno and G. Marchetto, "Process-Translatable Petri Nets for the Rapid Prototyping of Process Control Systems", IEEE Transactions on Software Engineering, Vol. SE-12, Feb. 1986, pp. 346-357.

[Brun86b] G. Bruno, P. Spiller, and I. Tota, "AISPE: an Advanced Industrial Software Production Environment", Proceedings of IEEE Computer Software and Applications Conference (COMPSAC), Chicago, Oct. 1986, pp. 94-99.

[Brun86c] G. Bruno and A. Elia, "Operational Specification of Process Control Systems: Execution of PROT Nets Using OPS5", Proceedings, 10th World IFIP Congress, Dublin, Sept. 1986, pp. 35-40.

[Bald88a] M. Baldessari and G. Bruno, "An Environment for Object-oriented Conceptual Programming Based on PROT Nets", in Advances in Petri Nets 1988, Springer-Verlag, 1988.

[Bald88b] M. Baldessari, V. Berti, and G. Bruno, "Object-oriented Conceptual Programming Based on PROT Nets", Proceedings of the International Conference on Computer Languages '"8, Miami, FL, Oct. 1988.

3.33 PSL/PSA -- Problem Statement Language/Problem Statement Analyzer

3.33.1 BACKGROUND

Synopsis

PSL/PSA is a computer-aided, structured technique for analysis and documentation of requirements and preparation of functional specifications for information processing systems. Through the Meta Systems toolset it has been interfaced with multiple analysis techniques and it provides guidelines and automated support for the development and management activities of the software process, as well as for maintenance and re-engineering.

PSL/PSA is based upon the use of a formal method of specification, the system description language called PSL, together with the analysis tool PSA. PSA provides the ability to record the system description in a central repository, analyze the objects and relationships, evaluate and modify the systems definition, and produce ad-hoc and predefined reports. It allows method-independent systems design and integration of the analysts work.

History

PSL/PSA was developed by the Information System Design and Optimization System (ISDOS) project in 1968 at the University of Michigan, directed by Dr. Daniel Teicherow. The work was assisted by an affiliated effort by Hasan Sayani at the University of Maryland. ISDOS was separated from the University of Michigan in 1983 as a commercial venture and is now Meta-Systems. PSL/PSA is currently provided and supported by Meta Systems and by multiple national and international distributers, including ASTEC. PSL/PSA was first used on a deliverable system in 1973.

3.33.2 DESCRIPTION

Problem Statement Language/Problem Statement Analyzer (PSL/PSA) provides an automated approach to systems requirements definition. The use of PSL/PSA does not depend on a particular model of the software process and can be customized to a variety of software development methods, such as DeMarco's Structured Analysis.

PSL is a non-procedural language intended for representing system requirements; it provides a textual representation of the logical system design. Founded upon entity-relationship modeling, PSL views a system in terms of objects and relationships. Objects have properties which have property values. Relationships are the connections or interrelations between the objects. It defines specific types of objects and relationships needed to capture the information necessary for functional requirements and specification. It provides representations for system input/output flow, system structure, data structure, data derivation, system size and volume, system dynamics, system properties, and project management.

PSA provides the ability to record the description of the system in a data base, modify it incrementally, perform analyses, and produce reports in a variety of forms. Its capabilities include analyzing the similarity of input and output, detecting gaps in the information flow or unused data objects, showing the dynamic behavior of the system, showing the objects, properties, and relationships from various views, and providing project management reports.

PSL/PSA can be incorporated into an integrated environment which customizes the software development process with the use of other methods and tools. When coupled with the Meta-Systems toolset, PSL/PSA supports and provides guidelines for the following activities of the software development process: requirements definition, system specification, system design, coding, documentation, and re-engineering. It also addresses the activities associated with project management, quality assurance, and maintenance, such as: risk analysis and complexity assessment, cost estimation, integration, and test planning and generation.

3.33.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

The vendor states that PSL/PSA is well-suited for use in the areas of embedded systems/process control, real time systems, scientific/engineering applications, systems programming, data processing/database systems, and large scale simulation/modeling. It has been used by more than 100 organizations in developing over 250 delivered systems, including real-time embedded systems, business systems, telephone switching systems, and weapons systems, as well as systems for air traffic control, satellite development, simulation, and image processing. The most frequently used languages have been Fortran, Ada, and CMS-2. It provides guidelines for large system development and has been used in developing small, medium, and large systems.

Target Constraints

The vendor states that the method is flexible enough to address target system requirements including timing constraints, spatial constraints, special features of the hardware architecture and the operating system, concurrency issues, fault-tolerance, and security of access and that it supports the modeling of machine-independent processes.

Modes of Expression (Tables 9,10)

PSL/PSA is based upon the use of the formal specification language PSL and provides automated support for specified documentation templates, narrative overviews of modules, structured English, PDL, and decision tables, as well as iconographical representation of finite-state diagrams, data-flow diagrams, control-flow diagrams, flowcharts, HIPO charts, and hierarchy charts. It provides guidelines for mapping the specification to simulation language and to PDL. The data for each phase is contained in an integrated central repository, which allows extraction of the desired view for the desired phase of the software process.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

The vendor states that PSL/PSA is well-suited for use with incremental development and may be used with behavior simulation and prototyping to clarify system requirements. PSL/PSA supports multiple analysis and review techniques.

Other Technical Aspects

PSL/PSA advocates strict up-front effort in the design phase in order to reduce coding and implementation time. The automated analysis and reporting features are intended to reduce the time spent in each phase.

Its automatic update capability reflects changes in requirement specifications throughout the database and it offers an automated ability to analyze the impact of all changes made to the system. Changes are made through the use of the Diagram Editor in Structured Arthitect, the Language Processor in PSL/PSA, and updates to Structured Architect - Integrator. PSA provides a date and time stamp capability for tracking modifications and produces database modification reports.

The re-engineering capabilities facilitate analysis, redesign, or documentation of an existing system, as well as bridging to new systems. PSA reports assist in identifying possible reusable components and the query capability allows examination of the compatibility of candidates for reuse. The ability of the central repository to support an object-oriented design also aids in reuse of portions of a design.

3.33.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

The method provides guidelines and automated support for activities involving risk assessment and cost estimation, project planning and scheduling, and configuration management. PSL/PSA maintains project information in the database. This involves identification of people involved and their responsibilities and schedules. Project management information is provided in the form of reports or ad hoc queries.

Communication Channels

Communication within the development team, between management and the development team, and between the client and the development organization is facilitated through the flexible reporting mechanism, which presents selected information for each given management level or communications situation. Graphic reports supplemented by text are designed to facilitate communication with the client. In addition to allowing the customer to specify the desired documentation content, the method's reporting system allows the use of graphic, matrix, list, and textual formats.

Quality Assurance (Tables 12,14,15)

PSL/PSA provides a framework and automated support for test planning and generation. The method assists in the early detection of inconsistencies and/or errors by allowing users to define model rules and then preventing users from breaking the model rules. It also provides for the use of ad hoc on-line queries to check for inconsistencies and errors.

The method has capabilities for tracking modifications and integrating the work of different analysts. It provides an integrated model which supports multiple phase cross-checking to ensure that consistency is maintained when changes are made to specifications, design, or code. The Problem Statement Analyzer concentrates initially on errors of comission, or inconsistencies, while it detects errors of omission at later checkpoints [Saya85].

The method provides specific directions for recording information concerning any trade-off studies, rationales, personnel, and changes related to specification/design decisions and maintains an automated record of options considered. Reports, analyses, and the query language may be used to confirm that the procedures of the method have been correctly completed.

Documentation Formats (Table 16)

PSL/PSA provides documentation templates which are tailorable and generated automatically based on data produced from other steps in the development process. These include requirements definition, functional, behavioral and architectural specifications, interface specifications, system structure charts, internal program documentation, test plans, and a change log, and other documents as required by Mil-Std 2167A.

3.33.5 EASE OF USE

Technology Insertion

In the vendor's opinion, the minimum qualifications needed by a development team leader for successful use of the method include a bachelor's degree, 1-2 years of development experience, a working knowledge of one programming language, and previous experience with at least one software system. An experienced developer should have an understanding of entity- relationship modeling and abstraction to use the method. A full range of training is available, including classroom and on-line tutorials, on-site consulting, on-line help, "hot line" service, and a users' support group.

It is estimated that a project manager would be able to acquire an understanding of the major features and benefits of the method in one day, and an experienced developer could learn the essentials of the method in ten days, while it would take three months for an experienced developer to achieve the level of expert user of the method.

Automated Facilities

The respondent states that PSL/PSA, coupled with the optional tooiset, provides a framework for customizing the software development process with the use of other methods and tools and can be incorporated into an integrated environment.

PSL/PSA may be combined with Meta Systems front-end tools, such as Structured Architect, to create dataflow diagrams, data dictionary entries, and process specifications which can then be translated to PSL. Structured Architect supports structured analysis, functional decomposition, requirements allocation and tracking. The work of different analysts is integrated with the Structured Architect - Integrator, which may be used to automatically generate a PSL/PSA database and provides support io the database administrator. QuickSpec provides user-friendly input and editing of information in the central repository.

Meta Systems back-end tools extend the capabilities to other development activities. Report Specification Interface customizes documentation, bridges to other tools, code, and PDL generation. The View Integration System performs data modeling and database design.

For additional details concerning automated support, refer to Tables 9, 10, 13, 14, 15, and 16.

3.33.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

PSL/PSA can be hosted on IBM mainframes under VM/CMS or MVS/TSO, or on Digital VAX/VMS.

Acquisition Costs

The vendor estimates the unit cost to acquire the method and required components to be between $3,500 and 55,000. The unit cost for other recommended components supplied by the vendor are between $12,000 and 16,000. The unit cost for technical training or a management overview is $1,500/day.

Contact Information

Meta Systems, Ltd.	(313) 663-6027
315 E. Eisenhower, Suite 200	
Ann Arbor, Michegan 48108	[Provider]

3.33.7 REFERENCES

[Saya85] Hasan Sayani, "PSL/PSA: New Generation Real-Time Extensions", Oct. 1985, pp. 223-246. (presented at National Conference and Workshop on Methodologies and Tools for Real-Time Systems, Washington, D.C.,Oct. 28-Nov. 1, 1985).

[Teic77] D. Teichroew and E. A. Hershey, III, "PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems", IEEE Transactions on Software Engineering, Vol. 3, No. 1, Jan. 1977, pp. 41-48.

3.34 **RM** -- Refinement Method

3.34.1 BACKGROUND

Synopsis

This method addresses the activities of design, coding, prototyping, and documentation. It is intended to provide ways by which the design may be gradually completed following a top-down approach while at the same time following the principles of object-oriented programming and information hiding. It results in programs which are structured as a hierarchy of layers. The method is in the public domain and available for general use.

History

RM is based on ideas found in stepwise refinement and object-oriented design. It was developed by Vaclav Rajlich who has published several articles on the method. The method was first used with respect to a deliverable system in 1986. Since this time several organizations have used close derivatives of RM to develop large systems.

3.34.2 DESCRIPTION

Refinement Method (RM) specifically addresses system design and implementation activities. It is founded upon the functional decomposition approach. The developer states that the concepts of stepwise refinement and module coupling/cohesion are essential to the method, which also is reported to incorporate object-oriented programming. RM is well-suited for use within the context of the incremental software process model; however, the most effective use of RM is not dependent upon any one process model. The steps of design and coding are merged into one step of "development". RM uses Ada as both program design language and programming language.

During the development, the program may be thought of as consisting of two parts: the existing part and the intended part. The development progresses as additions are made to the existing part and deletions made to the intended part. All undefined entities used in the existing part, but which are not yet defined, constitute the "backlog interface". Development is a series of repeating steps to define all entities and thereby remove them from the backlog interface, at which time the development is finished. The resulting program is structured into a hierarchy of layers.

The method also specifies how to test an incomplete system through prototyping of missing parts.

3.34.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

The developer considered RM well-suited for applications involving embedded systems or process control, scientific or engineering processing, systems programming, and large scale simulation or modeling. An example of a an application developed with the method is VIC/VIFDR, environments oriented towards software maintenance.

Less than five delivered systems have been developed using the method. However, several derivative and very closely related methods have come into use over the last three years, which this estimate does not take into account. Tl.ese derivative methods were used in large projects; RM is intended for use in large projects but has been actually used in medium-sized projects. The number of organizations that have used the method is less than five, and the most frequently-used implementation languages used to code systems developed with the method are Ada, C, and Pascal-VS.

Target Constraints

The developer stated that the method prescribes steps for handling concurrency issues; specific information was not provided.

Concerning portability issues, the lowest layers of software are configuration dependent. Porting means revision or rewriting of these layers. Hence, the size of the porting effort is always obvious.

Modes of Expression (Tables 9,10)

A number of textual and iconographical modes of representation are compatible with the method, which encourages use of hierarchy charts. No specific modes are required.

The method prescribes rules for mapping backlog interfaces or data-flow diagrams into the skeletons of modules, thus assisting in translating from one mode of expression to another. Across phases of the software process, the method guarantees consistency of specification and design with the code since specification and design are represented by certain layers of the code. Moreover, skeletons of the code are generated from backlog interfaces.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

Rapid prototyping and incremental or evolutionary development are required for clarifying system requirements or behavior.

Other Technical Aspects

The method assists in reducing the effort needed to incorporate changes in the requirements by producing layered software. The changes in the requirements propagate in a top-down direction through the layers, which reduces both the effort to make changes and the uncertainty of the changes. There are techniques in the method which limit the propagation of the changes.

Specification, design, and code are overlapped, with certain layers being specification-oriented, and others being design-oriented. The only documents separate from the code are backlog interfaces, and the consistency with the code can be checked automatically.

The method's layering principle for building software helps in identifying reusable components. The developer states that a scan of the backlog interfaces gives a good reuse assessment of the layers.

3.34.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

Guidelines are furnished for assessing complexity, tracking project progress, allocating development resources, and allocating personnel to tasks.

Communication Channels

Facilitation of communication within the development team is accomplished by the layering of the resulting program architecture and the backlog interfaces. Prototyping is an integral part of the method, which facilitates communication with management and with the client, who is involved with all prototyping phases throughout the life-cycle.

Quality Assurance (Tables 12,14,15)

Test planning is prescribed at one or more points in the method. Code walk-throughs are also required. The method provides for continuous prototyping of the missing parts of the software. This allows an early and continuous detection of inconsistencies.

Documentation Formats (Table 16)

The method provides fixed document formats for system structure chart, design document, and internal program documentation. These documents are generated based on data produced from other steps in the method.

3.34.5 EASE OF USE

Technology Insertion

Minimum qualifications for a development team leader include a bachelor's degree, no previous development experience, working knowledge of one programming language, and experience on one software system. These qualifications were given for successful use of the method.

Training is available in the form of hands-on demonstrations, overview presentations, classroom tutorials, and on-site consulting by the developer. To acquire a basic understanding of the major features of the method, a project manager would need five days. An experienced developer would require ten days to learn to use the method's essentials, and three months to achieve expert user level.

Automated Facilities

Not applicable.

3.34.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

Not applicable.

Acquisition Costs

A management overview costs $5,000. Technical training for single and low-volume users is $10,000, and $15,000 for high-volume users. The method is in the public domain.

Contact Information

Vaclav Rajlich 313-577-2477
Department of Computer Science
Wayne State University
Detroit, Michigan 48202 [Developer]

3.34.7 REFERENCES

[Rajl85a] V. Rajlich, "Stepwise Refinement Revisited", Journal of Systems and Software, March 1985, pp. 80-88.

[Rajl85b] V. Rajlich, "Paradigms for Design and Implementation in Ada", Communications of the ACM, Vol. 28, No. 7, July 1985, pp. 718-727.

[Rajl87] V. Rajlich, "Refinement Methodology for Ada", IEEE Transactions on Software Engineering, Vol. SE-13, No. 4, April 1987, pp. 472-478.

3.35 **SADT** -- Structured Analysis and Design Technique

3.35.1 BACKGROUND

Synopsis

SADT takes a data-flow view for analysis and design using a distinct notational scheme combined with an activity model. Project organizational requirements are also part of the method. The structured analysis part of the method uses pictograms to represent the properties of a system. Diagrams of interconnections and hierarchical arrangements of the figures comprise the system model. The design technique part of the method is a disciplined approach to system top level design, making use of the structured analysis effort. According to the chairman of SofTech [Ross85], the method is more effective addressing issues occurring in early and late stages of the system development process.

History

Hori's "cell model" theory [Hori72] provided the box notation, which covers a small percentage of the current method. The method has been evolving since the 1960's with the first use in the production of a deliverable system in 1974. It was first called a "system-blueprinting" method for documenting the architecture of large and complex systems. In 1977 SofTech laid proprietary claim to the design technique. More recently the SADT method has been interfaced with several other methods or techniques for detailed design and coding, including PSL/PSA, Yourdon, Jackson Structured Design, Warnier-Orr, HIPO, Nassi-Schneiderman, and Petri nets.

3.35.2 DESCRIPTION

Structured Analysis and Design Technique (SADT) is a systems oriented method addressing problem analysis, requirements definition, functional specification, and top-level design. Due to its age, the method has several extensions and variations. The primary usage of SADT has been as follows:

- Requirements, using models to understand present and future operations as well as for specifying and designing operational systems involving hardware, software, and people;
- Software systems, using models to define user requirements, identify system components and interfaces, and develop a top-level design;
- Project management, using models to make task assignments, define procedures, and analyze communications;
- Simulations, using models to analyze performance and man-machine interactions;
- Test planning and integration.

The model that is used to develop the requirements and top-level design becomes the basis of the documentation.

The overall approach is data-flow oriented and uses functional decomposition, as well as the programming practices of stepwise refinement, process abstraction, and abstract data types. The hierarchical structure allows one to change levels of detail when necessary.

The method's first part, structured analysis, makes use of box and arrow diagrams to depict system components. Included in the use of the diagrams are rules to manage complexity. The use of the diagrams is somewhat similar to the use of mathematical notation, as a representation of some object or system whose properties are based upon underlying assumptions. A collection of diagrams is called a model, and may be suitable for describing systems including or excluding any software subsystems.

The design technique part of the method makes use of the previous analysis part to develop specifications. One component of this is accomplished by a formal requirements language. SADT primarily addresses top-level system design. Ross has indicated that SADT is more effective in early and late stages of design but not as effective in detailed design of software systems. The same paper cites several applications of SADT which do not specifically involve software.

3.35.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

The method emphasizes system development rather than software development. Areas where the method is well suited include:

- Embedded systems or process control;
- Scientific or engineering applications;
- Expert systems or artificial intelligence;
- Systems programming;
- Large scale simulation or modeling;
- Distributed processing or network applications.

The method is best suited for medium to large projects but it has been used by 51 to 100 organizations in developing between 100 to 250 projects of all sizes. Specific applications have included telephone switching systems, corporate accounting systems, and aerospace manufacturing systems. The most frequently used languages have been Ada, Fortran, C, Jovial, and CMS-II.

Target Constraints

The vendor states that SADT addresses target constraints related to timing, space, concurrency, fault-tolerance, security of access, as well as specific features of the hardware architecture and operating system. It provides a graphical representation of these constraints, but does not contain specific syntax items for each. The method assists in porting end-product systems to different target configurations through the separation of functional analysis modeling from specific design models targeted to different configurations, as well as through the use of "plug-compatible" alternative decompositions for design elements.

Modes of Expression (Tables 9,10)

The aim of the method is to document all salient aspects of a system in a clear and concise way. SADT uses data and activity models to represent a system. There are two graphic forms, a design tree and a system model, where the system model is comprised of one or more activity charts. The box and arrow diagrams of SADT are used in the early and middle periods of design. They show both data-flow and control-flow in the same diagram. They are supplemented with hierarchical representations, narrative overviews and a formal specification language. Rules are prescribed for the mapping between the data-flow for the application and the support system interfaces. These rules involve a special notation for "inter-model ties".

Techniques for Analysis and Requirements Clarification (Tables 11,12)

SADT encourages the use of rapid prototyping, incremental development, simulation, dynamic animation, and executable specifications to clarify system requirements and requires the use of data-flow analysis, control-flow analysis, and design reviews.

Other Technical Aspects

The method assists in reducing the effort needed to fully incorporate changes in the requirements by isolating aspects, bounding the context, and providing top-down decomposition levels. Also, the concise graphics form displays the scope of the requirements.

It assists in ensuring that consistency is maintained among specification, design or code when changes are made to any of these through the required formal reader/author review cycle and the special notation for "mechanism" and "inter-model ties".

The vendor states that using "plug-compatible" modularity and "mechanism notation" reduces the effort required to reuse components. The packaging of reusable elements into sub-models, with rigorous interface modeling, assists in identifying possible reusable components.

3.35.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

SADT supports risk management and other project management activities, however it does not include details of how to perform them. Specific directions and procedures for project planning are prescribed, and the method provides guidelines or a framework for other activities. SADT sets forth the goals for the system, makes recommendations for allocating people and resources to different activities, and requires specific personnel roles.

Communication Channels

The method provides coordinated communication within the development team as well as between the relevant sectors of the development organization through the definition of formal review procedures. Design reviews and peer-review sessions make up part of this process. Top-down decomposition provides communication at various levels of detail. Graphic diagrams define interfaces and data components more concisely and rigorously than text.

Communication between management and the technical development team is facilitated through the use of levels of detail and support structures in graphic form, as well as multiple viewpoints, such as management and applications, of systems.

Communication between the client and the development organization is facilitated through the use of levels of top-down detail for requirements analysis models and high level design, graphic chart formats, and interface definitions to client operational environments. Moreover, there are special project models for management/client controls that are part of the proposal and project plan.

Quality Assurance (Tables 12,14,15)

SADT supports quality assurance activities, however it does not include details of how to perform them. It provides general guidelines for test planning and generation based on system requirements. It automatically generates material suitable for inclusion in the quality assurance or test plan from the data produced in other steps in the development process.

In order to ensure and assess conformity to specification, the method provides data-flow analysis, control-flow analysis and a formal specification language. It assists in ensuring that consistency is maintained among specification, design or code when changes are made to any of these by employing a special notation for "mechanism" and "inter-model ties". Reader/author peer reviews and formal design reviews also support the same efforts.

The method provides guidelines for configuration management. It gives specific directions for maintaining a traceable record of technical decision-making that include the specification/design options considered, trade-offs studied, rationale for any decisions, personnel involved, and other related changes.

Documentation Formats (Table 16)

Use of the method provides information in textual and graphical form suitable for inclusion in deliverable and internal project documentation. This information is automatically generated during the development process specified by the method.

3.35.5 EASE OF USE

Technology Insertion

Overview presentations, classroom tutorials and user manuals are available, as well as on-site consulting by the vendor and independent consultants. Third party publications describing the method are also available.

Project managers would be expected to take one day to acquire an understanding of the major features and benefits of the method, while it would take five days for an experienced developer to learn the basics of the method. It takes about two months to achieve the level of an expert.

To be able to use the method, a development team leader needs less than two years of college-level technical education, three to five years of development experience, a knowledge of one programming language, and experience with one different software system, as well as the ability to synthesize and be comfortable at various levels of detail.

Automated Facilities

The techniques required by SADT are incorporated in automated tools which support activities beyond the method alone. Tools such as DAFNE, IDEFine and SPECIF are used throughout the method. The graphics tools support the diagramatic aspects of the method.

The following CASE tools support modeling:

Product	Vendor	Hardware
Excellerator/DAFNE	Italsiel	IBM PC/AT
Design/IDEF	Meta Software	Apple
IDEFine-0	Wizdom Systems	IBM PC/AT
COINS	Eclectic Solutions	IBM PC/AT
SPECIF	Thomson/CSF	VAX and Apollo
AutoIDEF0	USAF (WPAFB)	Cyber
IDEF/Leverage	Dacom	IBM 43xx and VAX

For more information concerning the activities supported by automated tools, refer to tables 10 and 16.

3.35.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

The hardware/software configuration required is dependent upon the automated tools selected. The tools run on the hardware specified in the chart included in the discussion of automated facilities above. These include the IBM PC/AT, Apple, VAX, Apollo, Cyber, and IBM 43xx.

Acquisition Costs

Consulting rates range between $5,000 to $10,000 for single and low-volume users; for more than 15 users, the cost for the course and tools is $15,000. Unit costs for other recommended components supplied by the method vendor range between $6,000 to $12,000 per user for single and low-volume users, and between $4,000 to $8,000 per user for high-volume users. Unit cost for technical training is $1,200 per user for a minimum of six users, and $1,000 per user for more. A management overview costs $1,000.

SofTech has copyrighted the training materials. Licensing of the tools is based upon the policy of the vendor of the tool(s) selected.

Contact Information

Clarence Feldman — 617-890-6900
SofTech Inc.
460 Totten Pond Road
Waltham, MA 02254 — [Vendor representative]

Information regarding the various customer support tools and their vendors is available upon request from SofTech.

3.35.7 REFERENCES

[Hori72] S. Hori, CAM-I Long Range Planning Final Report for 1972, Chicago: Illinois Institute of Technology Research Institute, 1972.

[Ross85a] D. T. Ross, "Applications and Extensions of SADT", IEEE Computer, Vol. 18, No. 4, April 1985, pp. 25-34.

[Ross85b] D. T. Ross, "Douglas Ross Talks About Structured Analysis", IEEE Computer, Vol. 18, No. 7, July 1985, pp. 80-88.

3.36 SCR -- Software Cost Reduction

3.36.1 BACKGROUND

Synopsis

The SCR method has been chiefly associated with the concepts of information hiding and modularization by separation of concerns. The requirements document is divided into relatively independent parts in order to localize changes. Similarly, in system design, likely changes are encapsulated in one module or a small number of related modules to enhance ease of change.

The method is a product of a research effort in software engineering and is not for sale; however, basic tenets of the method are described in an extensive bibliography from the Naval Research Laboratory as well as in professional journals.

History

SCR has been developed over the past decade at the Naval Research Laboratory by a group of people including David L. Parnas, Paul C. Clements, Kathryn H. Britton, Stuart R. Faulk, and Bruce G. Labaw. The Naval Research Laboratory developed the procedures and documents of this method in connection with the redesign of the onboard software for the Navy's A-7E aircraft. Thus, the method is also called the NRL Software Engineering Methodology.

3.36.2 DESCRIPTION

SCR, Software Cost Reduction, is a method based upon the principles of information hiding and separation of concerns. The first principle is explained in [Clem84]: "Information-hiding [Parn72] is a method of designing software to minimize the impact (and hence, the cost) of making software changes. The method involves dividing the software into modules according to likely changes; each module is responsible for encapsulating or 'hiding' the effects of a change from the rest of the system. The key is to design the interface of each module so that it consists only of information about that particular module that is not likely to change. In that way, when changes that affect a module are required, only the implementation of that module is likely to require a change. The interface and all other modules that use the interface are not likely to change at all."

Separation of concerns is the principle whereby the design information is divided into distinct and relatively independent documents. Modularization by separation of concerns is tied to the concept of information hiding by requiring that when concerns are separated from each other, then the details about them be encapsulated in different modules, and thereby "hidden" [Hest81].

SCR provides a framework and model documents for most of the activities involved in software development, with the exception of risk or cost assessment. SCR was considered well-suited for use within the context of several software process paradigms, including the Waterfall model, Boehm's spiral model, rapid prototyping, and the incremental model. Essential to the method are the concepts of stepwise refinement, information hiding (as above), process abstraction, abstract data-types, and structured programming.

The products of the method's preliminary design phases are a set of documents which include:

- Requirements specification;
- Module decomposition;

- Module dependency;
- Process structure;
- Resource allocation;
- Module interface;
- Module design;
- Test plan.

The use of the method at NRL includes specific procedures that have been instituted to extend the method.

3.36.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

The developer rated the method well-suited for use on embedded systems or process control, time-critical or real-time applications, systems programming, and distributed processing or networks. Delivered systems in the area of embedded real-time avionics have been developed using the method; more specific information on numbers of systems developed with SCR or organizations having used the method was not supplied. The method is intended for use on projects of all sizes and has been used on small- and medium-sized projects. SCR is language independent.

Target Constraints

As for requirements of the target system, the SCR method does not prescribe specific steps for addressing such requirements so much as providing a template for documentation and an example to serve as a model. In particular, all timing requirements are defined in a specific chapter of the requirements document. The requirements are given in terms of mathematical functions for the system outputs. Functions may be periodic or demand. A timing constraint is associated with each function in terms of its period or deadline in real-time. Special features of the target hardware architecture and target operating system are given as part of the description of system inputs and outputs, since that is the part relevant to the software. Typically, this identifies all the essential characteristics such as timing, representation, and the instruction sequences for accessing particular devices.

Regarding concurrency issues, the specification is non-algorithmic by design. It does not specify an order among actions except where one is explicitly required and hence preserves the maximum concurrency. Fault tolerance issues are treated like any other system requirements. They must be expressed as constraints on the system outputs.

In general, the method is target and language independent. It can be tailored to specific features of the target architecture, operating system, and implementation language. The system is developed as a hierarchy of abstract machines. The abstract machine at a given level hides the target configuration. By re-developing the implementation of that abstract machine, e.g., a compiler, the system can be ported to a new configuration.

Modes of Expression (Tables 9,10)

Required textual modes of representation include formal specification languages, mathematical notation, and structured documentation. Narrative overviews of modules is considered to be inconsistent with the method. The method strongly encourages the use of finite-state diagrams, functions, and state tables to convey the representation iconographically. Although the modes of expression differ between documents, mappings by reference are given explicitly for tracing between documents. Across development phases, the method is seen as facilitating transformation by separating concerns. This allows the products of one phase to be broken into relatively independent parts which may then be assigned to an individual or team. It also helps by providing templates for the documentation produced for each stage as well as explicit, complete, and formal specifications to be used by the next phase.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

Both rapid prototyping and incremental or evolutionary development are strongly encouraged as means for clarifying system requirements. Design reviews are required and formal proof techniques are also strongly encouraged.

Other Technical Aspects

The method reduces the effort needed for requirements changes through systematic application of the principle of separation of concerns. First, the requirements document itself is divided into a set of relatively independent parts and written as formally as possible. Thus, changes to the requirements usually affect a small number of easily-identified parts of the requirements document. Changes can often be made by changing a few table entries.

Information hiding is applied in the system design specifically to enhance ease of change. The system modules encapsulate parts of the design that are likely to change so most changes result in changes to one module, or a small number of related modules.

The approach used by SCR helps ensure consistency first by not overspecifying the design at any stage. Thus, the requirements state only what the system must do and say nothing about its design. The module specifications give only the externally observable behavior and do not specify how the algorithms will be coded. The result is that most changes, e.g., choosing a different algorithm to implement a module, do not require a change to other parts of the specification and thus do not affect the consistency.

Where a change to one level of the design affects another, the changes are traced through sections of the documentation, giving the mappings between documents produced at each phase. Design/code reviews ensure that changes are correctly incorporated and the corresponding documentation updated.

In terms of assistance in identifying reusable components, the method develops the system as a member of a family of systems. The "uses" relation among system components is explicitly defined. The system modules are independent and embody such components as types, i.e., objects, and hence form the basis for reuse.

3.36.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

SCR requires that the activities of project planning, scheduling or manpower loading, allocation of personnel and development resources, and configuration management be addressed, but does not provide directions for accomplishing such. Other project management activities are not addressed.

The NRL uses UNIX RCS for configuration management.

Communication Channels

One of SCR's goals is to reduce the amount of communication required among members of the development team. This idea follows F. Brooks' assertion [Broo75] that adding personnel to a late software project will make it later, because each new member of the team must communicate with all the others. A primary goal of the information hiding/formal specification approach used in SCR is to reduce the communication required by breaking the software into relatively independent parts. Programmers design and code entirely from the formal specifications of the module interfaces and need not otherwise communicate with one another; everything a programmer needs to know or is allowed to know is defined on the module interface. These and the other system specifications are designed to facilitate communication between developers. In particular, the system Module Guide gives the overall structure of the system; the formal specification for each system module defines the required behavior.

The SCR approach is sometimes described as "design through documentation". The set of documents produced is designed to record all significant design decisions throughout the life-cycle. As such, one can determine from the completeness of the various documents how the development stands at a given time, thus facilitating communication between the development team and management.

Involvement of the software client occurs primarily during the development of the requirements document, which is designed to incorporate all system requirements in an implementation-independent specification of the required system. This document serves as the basis for defining what the client wants; it is what he reviews and approves. After that, he need not be involved in the process except where changes must be made; the requirements document also specifies which changes will be easy to make.

Thus, the requirements document serves as a vehicle for communication, can be used as the basis for contracting and testing, and is the arbiter for disputes. The method is consistent with but does not require rapid prototyping of the user interface.

Quality Assurance (Tables 12,14,15)

The method provides guidelines for test generation, unit/integration testing, generation of test data, and prescriptive checking of interfaces. Test planning and field or acceptance testing is required but no specific directions are provided for accomplishing such.

Maintaining a traceable record of technical decision-making during the software development process is required and the method provides a model document for such.

Documentation Formats (Table 16)

The majority of documents required by the method are tailorable in format, including a module guide. For specific documents, see Table 16.

3.36.5 EASE OF USE

Technology Insertion

The developer provided estimates of minimum qualifications needed by a development team leader for successful use of the method: a bachelor's or advanced degree, three to five years of development experience, knowledge of two or more programming languages, and experience on two or more different software systems. A master's level education in Computer Science or related field was considered best, as well as an understanding of finite state machines, formal languages, formal specifications, and relations and functions.

Means for training an organization in the use of the method include hands-on demonstrations, overview presentations, classroom tutorials, video tapes, user manuals, related publications from third-parties, periodic technical updates, model documentation, and technical papers.

Three to five days would be required for a project manager to acquire an understanding of the major features of SCR, and ten days for an experienced developer to learn to use the method's essentials. Such a developer would need three to six months to achieve expert user level.

Automated Facilities

NRL used UNIX with RCS for CM; in general, automated facilities are not applicable.

3.36.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

Not applicable.

Acquisition Costs

Not applicable.

Contact Information

U. S. Naval Research Laboratory 202-767-3212
Information Technology Division, Code 5575
Washington, DC 20375 [provider]

3.36.7 REFERENCES

[Clem84] P. C. Clements, R. A. Parker, D. L. Parnas, J. E. Shore, and K. H. Britton, "A Standard Organization for Specifying Abstract Interfaces", Naval Research Laboratory, Washington, DC, June 14, 1984.

[Hest81] Hester, D. L. Parnas, and D. F. Utter, "Using Documentation as a Software Design Medium", Bell System Technical Journal, Vol. 60, No. 8, Oct. 1981.

SCR

[Parn72] D. Parnas, "On the Criteria to be Used in Decomposing Systems into Modules", Communications of the ACM, Vol. 15, Dec. 1972.

[Parn78] D. Parnas, "Designing Software for Ease of Extension and Contraction", Proceedings of the 3rd International Conference on Software Engineering, The Computer Society Press, May 1978, pp. 184-195.

[Parn85] D. L. Parnas, P. C. Clements, and D. M. Weiss, "The Modular Structure of Complex Systems", IEEE Transactions on Software Engineering, Vol. SE-11, No. 3, March 1985.

An extensive bibliography on SCR and the A-7E program is available from the NRL.

3.37 SD -- Structured Design

3.37.1 BACKGROUND

Synopsis

Structured Design provides techniques to address the activities of preliminary design and detailed design. It is used to define the architectural ("structural") design of the system being developed - the individual program units and their interconnections, by using structure charts and data flow diagrams. It provides techniques for deriving alternate modular strucures and evaluating alternative decompositions.

History

The term "structured design" was introduced by IBM in [Stev74]. Prior to that, the various concepts were referred to as modular design, logical design, composite design, or the design of program structure [Pete81]. Stevens, Myers, and Constantine described the concepts and techniques of structured design, including structure charts, based on Constantine's research over the previous ten years [Stev74]. G. Myers provided an amplification of the underlying theory and examples [Myer78]. Based on DeMarco's work, the use of data flow diagrams was combined with structure charts.

Structured design was first used on a deliverable system in the early 1960's. Its use has been actively promoted by Yourdon, Inc. (now part of DeVry). SD provides the basis for many other methods. (See also ESA/ESD, Ward/Mellor, SADT, Stradis.)

3.37.2 DESCRIPTION

The techniques of Structured Design (SD) are used during the transition from user requirements to architectural design: the description resulting from structured analysis is refined to become the preliminary and detailed designs of functional modules. Structured design provides techniques to reduce the complexity of programs by dividing them into hierarchies of called functional modules. Data flow diagrams are used in conjunction with structure charts.

Structured design as presented by Yourdon and Constantine [Your86], [Your79] consists of:

- Documentation techniques - graphic and descriptive representations, including data flow diagrams and structure charts;
- Evaluation criteria and heuristics - guidelines for assessing a proposed structure, including the concepts of "coupling", "cohesion", "span of control", "scope of effect/scope of control" and "packaging";
- Design strategies - techniques for viewing the design requirements, including top-down design, transform-centered design, transaction analysis, and modular design [Parn72];
- Implementation strategies - plans for the sequence of coding and implementation.

Structured design starts with a system specification, generally in the form of data flow diagrams from structured analysis. From these diagrams, which show the inherent data flows and transformations, the natural aggregates are identified and structure charts are derived. The structure charts are defined, evaluated, and redefined iteratively, based on the "coupling" between and the functional "cohesion" within modules. There are three issues analyzed as coupling: interface complexity, type of connection, and type of communication. There are seven categories of cohesion analyzed. Finally, the system specification is ıe-examined and the process is selectively repeated.

While the method is founded upon functional decomposition, the data flow orientation differs from that of classical functional decomposition, where tree-like hierarchical structures are created based on top-down design. SD tends to retain the logical shape of the system, useful in identifying and organizing the physical design.

3.37.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

The method is best-suited for use in systems programming and business applications and is appropriate for other applications. It has been used in over 100 organizations for developing more than 250 delivered systems, including on-line and batch business applications and process control systems. The most frequently used languages have been COBOL, FORTRAN, PL/I, APL, Assembler, and Basic.

Target Constraints

The data flow orientation does not provide representation for the passage of time. The respondent stated that this simplifies the definition and user's review of transformations, input, and output. Other methods have built upon Structured Design and have added techniques to model timing concerns and other target constraints.

Modes of Expression (Tables 9,10)

Structured Design uses the structure chart, HIPO (Hierarchy, plus Input, Process, Output) charts, and narrative representations. The use of the data flow diagram was added, either provided as input to the structured design from previous structured analysis, or used within structured design, as a basis for deriving structure charts or to show additional levels of detail for the program modules being designed.

Alternate notations have been used by proponents of the various forms of structured design. These amount to different pictographs or icons representing the same objects or processes. For the most part, objects, structure and behavior are represented iconographically, instead of textually.

Structure charts are used to specify the modular characteristics of the software being designed with additional notation showing control flow. The IBM notation is consistent with the HIPO (Hierarchy, plus Input, Process, Output) technique. Yourdon and Constantine add notation including representation for macros, modules containing only data, operating system interface, looping, decisions, and automatic, asynchronous, or concurrent transfers of control. Their version also distinguishes between normal data and control indicators.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

The respondent stated that the method requires data structure analysis and control flow analysis while encouraging data flow analysis. Structured design requires that the requirements specification be completed prior to its use; then it provides a set oı rules to transform specifications into an architectural design.

Other Technical Aspects

SD is based upon the concepts of stepwise refinement, information hiding, and module coupling/cohesion. Its intent is to simplify the problem, reduce program complexity, and define interfaces in a manner that makes it easier to change. Functional decomposition is intended to isolate changes to small, independent modules. It is stated that the separation of input/output from logic modules reduces the effort required to port end-product systems to different target configurations. The respondent reported that the effort necessary to adapt end-product systems to new applications is reduced due to the degree of modularity that results from the use of this method. The method's requirement of a single implementation for each function assists in identifying possible reusable components.

The method's focus on measures for binding and coupling allows comparison of various alternative designs. The respondent stated that these criteria can be used to evaluate the designs resulting from the use of other methods as well.

3.37.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

The method provides specific directions and procedures for assessing complexity. The available evaluation checklists are designed to aid in managing and controlling the project.

Communication Channels

Data flow diagrams provide a representation which more closely resembles natural human values [Stev82]. The use of data flow diagrams and structure charts provides documentation for further development and maintenance activities. Definition of interfaces provides for communication between developers working on the separate modules of the system; developers need to communicate only when the interface changes. Management can monitor progress on the project by monitoring progress on individual modules.

Quality Assurance (Tables 12,14,15)

Structured Design is intended to reduce the original errors, simplify testing, and ensure the consistency of specification, design, and code by reducing complexity, defining interfaces, and using self-contained modules. It provides for data flow and control-flow analysis of dependencies. Formal definition of interfaces between modules is designed to cause the early detection of inconsistencies and/or errors. Evaluation checklists in [Stev81] are provided to confirm that the procedures have been correctly completed.

Documentation Formats (Table 16)

Structured Design provides a fixed representation for the architectural specification, the interface specification, the system structure chart, and the documentation of the design.

3.37.5 EASE OF USE

Technology Insertion

Training and consulting services are widely available in the form of overview presentations, classroom tutorials, on-site consulting by independent consultants, users' groups such as the Structured Methods Forum, and many reference books, textbooks, and articles.

The respondent stated that a project manager could learn the basics of the method in two days and a developer could learn the essentials in five days, while it would take eight months for an experienced developer to achieve the level of expert user of the method. In order to successfully use the method, a development team leader would require less than 2 years of college-level technical education, one or two years of development experience, a working knowledge of one programming language, and experience with two different software systems.

Automated Facilities

Many of the current CASE tools incorporate structured design and support activities beyond those addressed by the method alone. The respondent stated that the method can be incorporated into an integrated environment, and mentioned Excelerator by Intech and CASE2000 by Nastec as tools which support design call hierarchies.

3.37.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

The hardware/software requirements depend on the CASE tool selected, if any. A typical configuration would be a PC with 640K, a hard disk, EGA, and a mouse.

Acquisition Costs

Many companies and consultants provide courses and assistance in implementing the techniques of structured design with a variety of perspectives and alterations. Many reference books and articles are also available. Acquisition costs would be dependent upon the consulting services and tools selected.

Contact Information

Wayne Stevens 11 Myron Street Fairfield, CT 06430	203-259-2781 [Respondent]

3.37.7 REFERENCES

[Myer75] G. J. Myers, Reliable Software through Composite Design. New York: Petrocelli/Charter, 1975.

[Stev85] W. P. Stevens, "Using Data Flow for Application Development", Byte, June 1985.

[Stev82] W. P. Stevens, "How Data Flow Can Improve Application Development Productivity", IBM Systems Journal, Vol. 21, No. 2, 1982, pp. 162-178. (Reprint Order No.G321-5165).

[Stev81] W. P. Stevens, Using Structured Design. New York:John Wiley and Sons, 1981.

[Stev74] W. P. Stevens, G. J. Myers, and L. L. Constantine, "Structured Design", IBM Systems Journal, Vol. 13, No. 2, May 1974, pp. 115-139.

[Your79] E. Yourdon and L. Constantine, Structured Design, New Jersey: Prentice Hall, 1979.

3.38 SEM -- System Engineering Methodology

3.38.1 BACKGROUND

Synopsis

The method is a collection of techniques for developing "effective, reliable, and useful software-oriented systems for a wide range of application domains (business, industrial, and military)" [Wall87]. The following development aspects are addressed:

- Understanding the system requirements through structured analysis modeling methods;
- Developing complete, verifiable specifications;
- Evaluating proposed system behavior through prototyping and simulation;
- Developing architectural-level designs for software;
- Providing needed information to development personnel.

History

The origins of the method are in Structured Analysis and Design Technique (SADT), the Software Cost Reduction (SCM) project and Systems Analysis of Integrated Networks of Tasks (SAINT). Components of the method have been in use since 1975; the complete method was first used in 1981. The developers of this method are Robert Wallace and John Stockenberg.

3.38.2 DESCRIPTION

The Systems Engineering Methodology (SEM) consists of a number of methods and tools. The parts of the software process addressed with specific procedures by the method are requirements definition, system specification, system design and interface definition.

The system requirements are analyzed with modeling methods based upon the U. S. Air Force's Integrated Computer-Aided Manufacturing (ICAM) Definitional Methods and structured analysis techniques. Specifications are analyzed by applying the U. S. Naval Research Laboratory's Software Cost Reduction (SCR) Methods. SCR methods are also used to develop the architectural design for software. The proposed system is evaluated through prototyping making use of the U. S. Air Force's Systems Analysis of Integrated Networks of Tasks (SAINT) simulation language.

A technique known as Active Integration of Information provides a way of integrating all of the parts cited above. It provides a framework for generating, managing, distributing and communicating information. Guidance is provided on how individual steps are to be carried out, describing required information, information relationships and what information is needed next.

3.38.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

The method's developer rated SEM as being well-suited for use with applications involving embedded systems cr process control, time-critical or real-time processing, distributed processing or networks, expert systems or artificial intelligence, and large scale simulation or modeling. SEM was used to develop the Trident

Defensive Weapons System/Combat Subsystem and the SAM Missile Simulator. The complete method has been used on less than five delivered systems within the same number of organizations; however, portions of the method have been much more widely used. The method is intended for use on medium and large projects, asnd has been used for projects of both sizes. It has been used with most high-level implementation languages as well as with macro assembler languages.

Target Constraints

Timing and resource constraints are expressed in terms of control flow (behavior) models. Simulation is used to analyze system timing. Simulation can address multiple processors, channels, devices, and contention; modeling and simulation are specifically oriented towards concurrency issues. To the extent that simulation and modeling can deal with additions and deletions of resources and functionality, the method can also address fault-tolerance issues.

With regard to portability, the method is primarily concerned with front-end system definition. It does insure that any system description is as independent from implementation as possible.

Modes of Expression (Tables 9,10)

Textual modes of representation required by the method are specified documentation templates, mathematical notation, and decision tables. Required iconographical modes include data-flow diagrams, control-flow diagrams, and entity-relationship diagrams. The method provides automated support for all these modes as well as for other modes of representation which are strongly encouraged or compatible with the method. Flowcharts, HIPO charts, and Nassi-Shneiderman charts were considered inconsistent with SEM.

The method prescribes mapping rules for translating from data-flow diagrams to control flow diagrams or petri nets, and from data-flow diagrams and control flow diagrams to template based specification.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

SEM requires rapid prototyping and simulation in order to clarify system requirements or behavior. Analysis and review techniques include data-flow analysis, control-flow analysis, decision tables, author/reader cycle, and active reviews. Design reviews are strongly encouraged.

Other Technical Aspects

The developer stated that separation of concerns and the level of granularity insures that only one place in the requirements needs changing for each proposed system change. The method assists in ensuring consistency between entities of the development process by providing a common semantic base to all phases so that traceability is built into the development process. Consequently, there is no ambiguity in tracing from high-level requirements to specifications and design. The use of separation of concerns and information hiding assists in developing reusable specification and design components.

3.38.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

SEM prescribes specific directions for estimating initial cost and project planning. Guidelines are provided for accomplishing other project management activities; see Table 13.

Communication Channels

The developer stated that the intuitive nature of SADT/IDEFO, data-flow diagrams, graphical control-flow diagrams, and incremental review procedures were all designed to facilitate and coordinate communication within the development team. The method is designed to facilitate communication between the technical development team and management with its project planning facilities and granularity rules that allow accurate cost and size estimates and allow progress to be monitored. With regard to client and the development organization, client reviews are facilitated with user-oriented high-level description techniques (data-flow, control-flow, information modeling). These techniques can even allow the client to participate in system definition. Also, the method provides for traceability to the final product.

Quality Assurance (Tables 12,14,15)

SEM provides a framework for conducting a number of testing activities. The method assists in early detection of inconsistencies by using graphical language to define systems functions and behavior. There are built-in review procedures and simulation for dynamic feedback as well. In addition, the method provides specific directions for recording information regarding specification/design options which were considered, trade-off studies, rationale for any decision, and personnel who were involved in making a decision.

Documentation Formats (Table 16)

The documents required to be produced by the method are fixed-format when they are automatically generated, and tailorable in format when they are produced from user responses to computer-directed prompts. See Table 16.

3.38.5 EASE OF USE

Technology Insertion

The developer estimated that a development team leader would need a bachelor's degree, three to five years of development experience, knowledge of one programming language, and experience on one software system in order to successfully use the method.

Training is provided in overview presentations, classroom tutorials, on-site consulting by the vendor or independent consultants, user manuals, and related publications from third-parties. Two days would be requried for a project manager to acquire an understanding of the major features and benefits of SEM. Ten days would be required for an experienced developer to learn to use the essentials, and two to four months to become an expert user.

Automated Facilities

The developer listed Design/IDEF by Meto Software as supporting most of the activities of the method, and SAINT by Wright-Patterson AMRL to support simulation. See also Tables 9, 10, and 16.

3.38.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

Appropriate host configurations are networked MacIntosh PCs. SUN/UNIX and IBM/PS implementations are under development.

Acquisition Costs

Cost to acquire the method and required components is $5,000 for a single user, $4,000 for low-volume users, and $3,000 for high-volume users. Technical training is $1,000 for low-volume users and $750 for high-volume users. A management overview costs $700 except for high-volume users, in which case the cost is $300.

There is a licensing policy: a workstation license is approximately $2,000, and a site license is approximately $10,000.

Contact Information

John E. Stockenberg
30 East St.
Newport, RI 02840

401-847-8875

[Provider]

3.38.7 REFERENCES

[Wall87] R. H. Wallace, J. E. Stockenberg, and R. N. Charette, A Unified Methodology for Developing Systems. Intertext Publications, New York: McGraw-Hill, 1987.

3.39 SSD -- Hatley/Pirbhai - Strategies for System Development

3.39.1 BACKGROUND

Synopsis

In the Hatley/Pirbhai method, structured analysis techniques have been tailored to the development of real-time systems. The method focuses primarily on activities of requirements definition and specification definition. It extends the self-consistency checking of structured analysis to the system as a total entity, not only to the software component of the system.

The method is available for general use.

History

The method was developed by Derek Hatley and Imtiaz A. Pirbhai. It is based on the concepts of structured analysis and the theory of finite-state machines. The information hiding concept of D. Parnas has also influenced the development of this method. The method was first used with respect to a deliverable system in 1982.

3.39.2 DESCRIPTION

The Hatley/Pirbhai Strategies for System Development, or SSD, focuses on the definition of requirements and specification for the total system, not only the software component of the system. This goal is attained by building two models of the system, the system requirements model and the system architecture model. In the system requirements model, a statement of the problem is formulated which is independent of technology. Thus, a thorough understanding of the problem can be gained without deciding on a particular solution. In the systems architecture model, a technology-dependent solution is constructed in order to specify how the problem is to be solved with available technology. This process of defining system requirements and system architecture can be repeated at a lower level to model both the hardware components and the software components of the system.

In building the system requirements model, a data flow diagram (DFD) is used to construct a process model which provides a decomposition of the system's functional requirements. In this model, related functions are grouped together, unrelated functions are separated, and each function is specified non-redundantly. A data flow diagram is non-procedural and the processes within a DFD are non-hierarchical. In this idealized DFD model of the system, a process is assumed to operate instantaneously, triggered once sufficient data is available to do its task.

The next step requires the construction of a control model in order to define control flow through the system's functions, and to define finite state behavior processing. The finite state projection provides information on the processing of discrete signals, and how these signals trigger different modes of behavior for the system. Data and control are kept separate by the creation of control flow diagrams which parallel the data flow diagrams. State transition tables, decision tables, and activation tables are also used to further clarify the requirements.

The requirements dictionary is the principal tool for ensuring a formal and rigorous approach. Contained in the requirements dictionary is an alphabetical list of data and control flow names and a definition of each in terms of its components and structure. The dictionary must contain every data flow name and data store name. Group names must be decomposed into precise components, and ultimately everything must be broken down to primitive physical entities. The dictionary may contain attributes for these primitive entities, including units, range, or accuracy.

The system architecture model is built based upon the requirements model. This model addresses the following issues:

- What are the physical components of the system?
- What tasks are to be performed by each of the physical components?
- How do these physical entities communicate with each other?
- What are the communications among these physical entities?

The steps taken in creating the architecture model are: 1) to enhance the requirements model by using a specified template; 2) to define those processes concerned with user interface, input/output, maintenance and self-test; and 3) to allocate the requirements model to architecture modules. During each of these activities, an evaluation of the allocation is made, and trade-offs are decided.

The system architecture model components include:

- Architecture context diagrams which are used to show the communication between the system and entities in the environment in which the system exists;
- Architecture flow diagrams which map a group of data/control flows and processes from the requirements model into architecture modules;
- Architecture interconnect diagrams which represent channels by which the modules communicate;
- Architecture module specifications which describe the functions of individual modules, and establish traceability between the requirements model and the architecture model;
- Architecture interconnect specifications which establish the characteristics of the interconnect channels between modules;
- The architecture dictionary which establishes the allocation of data and control flows from the requirements model to the architecture model.

The Hatley/Pirbhai method emphasizes the need for a thorough understanding of the requirements of a system, and for a specification of the system which is traceable to requirements.

3.39.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

SSD is considered suitable for use especially in applications involving embedded systems, process control, real-time constraints, scientific or systems programming, distributed processing, and large scale simulation or modeling. Examples of specific types of applications for which delivered systems have been developed using the method are: avionics systems, process control, instrumentation, communications, test equipment, computer peripherals, medical systems, and consumer electronics.

It is estimated that between 21 and 100 systems have been developed, with about 51 to 100 organizations, or divisions within large corporations, having used the method. The method is intended for use on projects of all sizes, and has been used for projects of all sizes.

The implementation languages most frequently used when coding systems developed with this method are FORTRAN, Jovial, Pascal, Ada, Objective C, C++, and Smalltalk.

Target Constraints

SSD prescribes steps for handling a number of target system requirements. A timing specification is part of the method, which tabulates input-to-output timing relationships. The module specifications may contain information on spatial constraints. Both the module and channel specifications address special features of the target hardware architecture, and special features of the target operating system can be included as architectural constraints. Concurrency issues are incorporated inherently in the method. Fault-tolerance issues specifically are addressed by module and channel redundancy, and security of access considerations could be included in the functional requirements.

Portability concerns are addressed by the method by specifying the system as configuration independent, then mapping into any configuration through requirements/architecture transform.

Modes of Expression (Tables 9,10)

Required textual modes of representation include specified documentation templates, narrative overviews of modules, and decision tables. Required iconographical modes include finite-state diagrams, data-flow diagrams, and control-flow diagrams.

The method prescribes mapping rules for translating between the requirements model and the architecture model. This is an essential part of the method. The lowest level architecture can be specified using a PDL leading directly to code.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

SSD uses incremental or evolutionary development as a means of clarifying system requirements. Analysis techniques that are required are data- structure, data-flow, control-flow, and decision tables.

Other Technical Aspects

The developer reports that the method is non-redundant and captures all relationships. For these reasons the location and impact of change is made clear, thereby assisting in the effort to incorporate changes in the requirements. The specification and design are tightly linked through requirements and architecture models, thereby ensuring that consistency is maintained between the two, as well as with the code. In addition, the method contains self-consistency checks to assist in early detection of inconsistencies or errors.

3.39.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

The developer stated that SSD can aid in accomplishing activities associated with project management.

Communication Channels

Specific aspects of the method are designed to facilitate and coordinate communication between all parties. These are a graphical presentation, a hierarchical approach which allows selection of levels of detail, and separation of concerns: process, control, data, and architecture.

Quality Assurance (Tables 12,14,15)

SSD does not address quality assurance issues. However, design reviews, code walk-throughs, and Change Control Board reviews are encouraged. In addition, there are a number of tools which have provisions for maintaining a traceable record of technical decision-making.

Documentation Formats (Table 16)

The documentation required by the method is tailorable in format. The amount of automated support for generating the documentation varies depending on the characteristics of the different tools supporting the method.

3.39.5 EASE OF USE

Technology Insertion

The developer estimated that a team leader would need a minimum of 1 to 2 years development experience and a Bachelor's degree for successfully using the method. Because the method is not software dependent, no knowledge of programming languages is necessary, nor is experience with different software systems needed. However, an experienced developer should understand the theoretical constructs of finite state machines, data flow analysis, and BNF notation in order to use the method successfully.

Training assistance is available in the form of overview presentations, classroom tutorials, consulting by vendors and independent consultants, video tapes, and a text written by the developers. Learning time estimates were one day for a project manager to understand the major features and benefits, 5 days for an experienced developer to use the essentials, and 2 months for an experienced developer to become an expert user.

Automated Facilities

The developer makes a distinction between the method and the automated support associated with the method. There are many tools which support the method and which differ in their characteristics. The following tools and their vendors were listed which support the requirements model of SSD:

Teamwork	Cadre Technologies
Casetools	Mentor Graphics
Software through Pictures	IDE
Power Tools	Iconix
Excelerator	Index Technology
Autocode	Integrated Systems
Designaid	Nastec
Promod	Promod

VS Designer	Visual Software
Adagen	Mark V Systems

The developer noted that several vendors are considering adding the architecture model to their tools.

3.39.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

Automated tools supporting the method would appropriately be hosted on a VAX Station, a SUN or Apollo Workstation, or marginally on a PC.

Acquisition Costs

The method is available publically with no licensing policy in effect. Training costs vary by consultant.

Contact Information

Derek J. Hatley — 616-241-8832
Smiths Industries
4141 Eastern Ave., SE
Grand Rapids, Michigan 49518-8727 — [Developer and provider]

Imtiaz A. Pirbhai — 206-324-4137
Systems Methods
2026 Yale Ave., E
Seattle, Washington 98102 — [Developer and provider]

3.39.7 REFERENCES

[Hatl87] D. J. Hatley and I. A. Pirbhai, Strategies for Real-Time System Specification. New York, NY: Dorset House Publishing Co., 1987.

[Hatl85] D. J. Hatley, "A Structured Analysis Method for Real-Time Systems", presented at the Fall DECUS U. S. Symposium, December 1985.

3.40 STATEMATE

3.40.1 BACKGROUND

Synopsis

This commercially available method is primarily based upon visual representations and languages. The vendor states that the method is a comprehensive tool for specification, design and analysis of large, complex systems. Specifications are dealt with by means of visual languages called the Statechart, Activity-chart, and Module-chart languages. Along with editing, analysis and management support tools, there is a simulation capability available for testing a design, thereby supporting rapid prototyping of the target system.

History

The method extends the state/event paradigm which involves the use of state-transition diagrams. It also extends concepts related to data-flow analysis and finite state machines. Two principal developers are Drs. A. Pnueli and D. Harel. The method was first used with respect to a deliverable system in 1983.

3.40.2 DESCRIPTION

STATEMATE is a complete environment for specification, design and analysis of real-time systems. The development activities addressed with specific directions by the method are requirements definition/clarification, system specification, software quality assurance, and validation of specification or design through simulations, executions, dynamic testing and rapid prototyping. The method is well-suited to a variety of software process paradigms. The vendor considers STATEMATE to be compatible with a number of approaches to software development and with a number of programming practices. The method supports the specification and analysis of systems, including both hardware and software.

The vendor uses the term "reactive" systems when describing suitable applications for the method. This term refers to " ... systems whose dynamic behavior is governed by complex interactions both among subsystems and between subsystems and their environments." Examples of such systems are: real-time computer embedded systems, interactive software systems, integrated circuits, and control and communication networks.

A conceptual model is used by the method to deal with notions, entities and procedures relevant to reactive system development. Three visual languages are included in the supporting collection of software tools. Using icons to represent these visual languages, in an interactive setting, a software developer establishes a specification of the behavioral, functional and structural aspects of the system.

STATEMATE is said by its developer to be capable of semantic analysis of the description of a system from the three different viewpoints outlined above. Behavioral specifications are represented with "statecharts", a creation of one of the developers, D. Harel. Statecharts extend the concept of finite state machines by using AND/OR logic to support the decomposition of states, resulting in a visual representation that is said to overcome the limitations of state-transition diagrams. Statecharts are capable of addressing issues such as: specifying "behavior hierarchically"; concurrency - both synchronous and asynchronous; and basing the current system behavior on its past behavior.

In addition to statecharts and the statechart language, there are two others, the activity-chart language and the module-chart language. The activity-chart language is used to specify the functionality of the system while the module-chart language is used to specify the structure of the system. Documents and reports are generated throughout the development process.

Analysis and simulation tools are used to check consistency, completeness and the correctness of the specification from the viewpoint of the client. During interactive execution the user can specify the environment and then see the dynamic behavior which results on the screen.

3.40.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

STATEMATE was created to handle applications involving complex, control-oriented real-time systems. Its first use was on the Lavi mission-adaptive avionics package produced by Israeli Aircraft Industires. Other types of applications for which delivered systems have been developed with STATEMATE include software process modeling to improve logistics of documentation support for a large military system, specification and design of a new version of a military aircraft as well as of a control system, specification and validation of a communications protocol, and conceptual analysis of large distributed systems. The method was intended for use on medium and large-sized projects, and has been used on projects of all sizes. An estimated 51-100 organizations have used STATEMATE, and between 5-20 delivered systems have been developed with the method.

Target Constraints

STATEMATE incorporates timing constraints in the statechart language and addresses these constraints by means of dynamic testing. Concurrency issues are addressed in the same way. Special features of the target hardware architecture and the target operating system can be addressed by running the specification in the target environment via the Prototyper module with its code generation capabilities. In addition, fault-tolerance issues can be modeled and tested, and security of access can be analyzed using the model.

The method addresses portability issues by allowing the software developer to explore different architectural configurations using module-charts.

Modes of Expression (Tables 9,10)

The method provides automated support for a variety of textual modes either strongly encouraged by or compatible with the method. It requires three iconographical modes of representation: 1) Statecharts, an extension of finite state machines, 2) Activity-charts, which are similar to data-flow diagrams and 3) Module-charts, which show the physical structure of the system. All three modes are provided with automated support by the method and associated with the method's three languages of the same names.

Among the above three modes are mappings for translating from one mode to another. Mapping rules are prescribed to go from control activities within the activity chart to statecharts, from activities to modules, and from modules to activity-charts. The vendor considers that the method facilitates the transformation across phases of the software process by supporting several conceptual levels of development with the same model, as well as by reports, executable models and their databases, and test suites.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

The method strongly encourages a number of techniques to clarify system requirements. Analysis and review techniques are also encouraged, including interactive or batch simulations, static analysis for consistency and completeness, and dynamic analysis, e.g., reachability, deadlocks, non-determinism.

Other Technical Aspects

STATEMATE assists in the early detection of inconsistencies and/or errors by allowing a model of the system to be created, syntactically analyzed, executed, dynamically tested, prototyped, and debugged. Consistency is maintained at different phases of development through test scenarios and the test results which are defined and achieved. The method also encourages the identification of units that can be treated as reusable components.

3.40.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

Automated support is provided by the method to assess feasibility of system or application, for configuration management, and for reliability estimation.

Communication Channels

The vendor states that the clarity and precision of the STATEMATE model facilitate communication within the development team. Also helpful are test scenarios used during simulation and the prototype code that can be generated directly from the database.

To facilitate communication between the technical development team and management, the STATEMATE specification can be executed and prototyped, with proof of concepts shown in an animated execution or prototyped form. Also, verified pieces of the system and how they fit together into the system as a whole can be shown. In addition to the above, communication between the development organization and the client is aided by the Prototyper module, which generates code from the STATEMATE specification, allowing the specification to be run in the target environment for further evaluation by both parties. This can be done at any stage of the project. The code can be linked to a soft panel, i.e., graphic representation of the user interface, allowing users to try out the prototype as if it were the real system.

Quality Assurance (Tables 12,14,15)

The method provides procedures for conducting test planning at one or more points in the software process, generation of tests, prescriptive checking of interfaces, and dynamic testing of the specification. These procedures have automated support. The method also provides a specific and automated level of configuration management.

Documentation Formats (Table 16)

The format of almost all documents required by STATEMATE is fixed and generated based on data produced from other steps in the method. However, the user can tailor his reports to include a combination of text and graphics. A quality assurance/test plan document is not automated and has a tailorable format.

3.40.5 EASE OF USE

Technology Insertion

Minimum qualifications needed by a development team leader in order to successfully use the method include a bachelor's degree, no previous development experience, working knowledge of one programming language, and experience with one software system. State machine knowledge is also helpful.

Training assistance includes hands-on demonstrations, overview presentations, classroom tutorials, on-site consulting, video tapes, a "hot line" service, user manuals, a users' support group, related publications from third parties, and periodic technical updates. One day would be required for a project manager to acquire an understanding of the major features of the method, and two days for an experienced developer of five or more years' practice to learn to use the method's essentials. Three months would be required for such a developer to become an expert user.

Automated Facilities

The method provides automated support for a number of representation modes, generation of required documents, testing activities, and project management activities. See Tables 9, 10, 13, 15, and 16.

3.40.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

The following configurations were reported as appropriate for hosting the method:

- VAXStation/MicroVMS 4.7 or higher/UIS 3.2 or higher
- Apollo Series 3000 or 4000/Aegis 9.7/Domain 1X 9.6/GMR
- Sun-3/SunOS 3.4/NeWS 1.1

Acquisition Costs

Cost to acquire the method and required components is $10,000. Technical training is included. The Analyzer tool is recommended and costs $25,000. Optional tools are the Prototyper at $25,000 and the Documentor at $15,000. Volume discounts are available and vary with package. There is a licensing policy which covers the maximum number of simultaneously active users per network. A site license is available.

STATEMATE

Contact Information

i-Logix Inc. 22 Third Avenue Burlington, MA 01803	617-272-8090 [provider]

3.40.7 REFERENCES

[Hare87a] D. Harel, "Statecharts: A Visual Formalism for Complex Systems", Science of Computer Programming, 8.3, June 1987, pp. 231-274.

[Hare87b] D. Harel, A. Pnueli, J. P. Schmidt, and R. Sherman, "On the Formal Semantics of Statecharts", Proceedings of the 2nd IEEE Symposium on Logic in Computer Science, Ithaca, NY, June 22-24, 1987, published by IEEE Press, NY, 1987, pp. 54-64.

[Hare88] D. Harel et al, "STATEMATE: A Working Environment for the Development of Complex Reactive Systems", Proceedings of the Tenth IEEE Conference on Software Engineeεring, Singapore, April 13-15, 1988, published by IEEE Press, NY, 1988, pp. 396-406.

3.41 StP -- Software through Pictures

3.41.1 BACKGROUND

This method provides an integrated graphical environment targeted to support the analysis and design phases of software development. It allows the user to customize the environment, to be able to use a variety of well-known methods, and to generate comprehensive template-driven documentation automatically. StP provides complete integration of information through a multiuser data dictionary built on a relational DBMS.

Closelv associated with this environment is the User Software Engineering Methodology which is aimed at the development of interactive information systems. StP is available for general use.

History

This method was developed at Interactive Development Environments (IDE) by A. Wasserman and P. Pircher. It supports the User Software Engineering Methodology of IDE, development of which was begun in 1975. StP was first used for a deliverable system in 1984. The method also incorporates concepts associated with the methods of T. DeMarco, E. Yourdon, L. Constantine, C. Gane, T. Sarson, D. Hatley, I. Pirbhai, M. Jackson, and P. Chen.

3.41.2 DESCRIPTION

Software through Pictures (StP) is an integrated environment which supports a variety of graphical tools for analysis and design. StP addresses requirements definition, system specification, and system design. It is founded upon data-flow and control-oriented approaches, *as well as on entity-relationship modeling.* Its best use is not linked to any particular paradigm of the software process. StP allows users to model software systems using any of the following techniques:

- Data-flow diagrams using either Gane/Sarson or DeMarco/Yourdon symbols;
- Structure charts using the Yourdon/Constantine representation;
- Entity-relationship diagrams of Chen;
- Data structure representations of Jackson;
- Real-time representations using the method of Hatley/Pirbhai;
- User Software Engineering techniques of Wasserman for rapid prototyping of user interfaces.

Additionally, StP provides support for the software development process through the following tools:

- PICture: an object-oriented drawing tcol;
- Troll/USE: a relational database which contains the common data dictionary used by StP, but which may also be used by the developer as a DBMS;
- Document Preparation System: an auxillary tool that supports the generation of documents from the StP Data Dictionary to a variety of target output languages and devices;
- A collection of user-callable routines in the IDE tools library and the IDE object management library, including typesetting support, general utility routines, and multi-user project support which includes complete diagram locking and version control.

Finally, StP is itself based on an open architecture, and provides the user with all file formats, the format of the data dictionary, and numerous other features. Thus, the user can extend and customize the development environment.

An associated method developed at IDE, User Software Engineering (USE), focuses on interactive information systems. The following goals guided its development:

- Functionality: cover the entire process of developing a system with a predefined set of requirements;
- Reliability: support the creation of systems which do not inconvenience users by system crashes, loss of data, or lack of availability;
- Usability: assist the developer in assuring, as early as possible, that the resulting system will be easy to learn and easy to use;
- Evolvability: encourage documentation and system structuring so that the resulting system is easily modifiable;
- Automated support: provide automated tools that improve both the process of software development and the resulting system;
- Improved developer productivity: reduce the time required to create a properly functioning system;
- Resuability: provide a method which is itself reusable for a large class of projects, and provide design products which are themselves reusable on similar future projects.

To achieve these goals, the USE method follows an evolutionary approach in which a sequence of prototype systems is developed by following a well-defined set of phases. The four aspects of the analysis phase involve data modeling, activity modeling, analysis of user characteristics, and analysis of usage characteristics. During the external design phase, the method, like many other approaches, uses data abstraction and data modeling. But rather than using a traditional "top-down" approach to refine system functions, USE follows an "outside in" approach in which transition diagrams are employed to model the external interface.

The developer believes that in both the analysis and the design phases, it is useful to be able to employ a variety of techniques. StP supports this capability.

[Authors' note: Another method, OOSD, or Object-Oriented Structured Design, was recently developed at IDE by A. Wasserman, P. Pircher, and R. Muller. Due to publishing deadlines, a description of this method could not be provided. However, Section 3.41.7 includes a reference [Wass89] on OOSD].

3.41.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

The developer reports that StP is well-suited to a number of different application areas. Examples of delivered systems built with this method are voice mail, commodities trading , radar systems, and clinical laboratory equipment. Between 100 to 250 systems have been developed with the method, within more than 100 organizations. The method is intended for all project sizes and has been used as such. Most frequent implementation languages used when coding systems are C, Ada, and Pascal.

Target Constraints

Target constraints are addressed by annotation types. There are annotation types for timing of processes, for memory requirements of processes and modules, for special features of the target hardware architecture and target operating system, and for performance requirements.

Modes of Expression (Tables 9,10)

No modes of expression are required by the method. Instead, several textual and iconographical modes are strongly encouraged. The method encourages specified documentation templates, narrative overviews of modules, Structured English, PDL, and decision tables. Encouraged iconographical modes include finite-state diagrams, data-flow diagrams, control-flow diagrams, and entity-relationsbip diagrams. Automated facilities for all the above modes are provided within the method.

The method facilitates transformation across phases of the software process by the fact that PDL and code templates are associated with the Structure Chart Editor and the templates are filled automatically from the StP Data Dictionary.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

Rapid prototyping is strongly encouraged for clarification of system requirements. Encouraged analysis and review techniques include data-structure, data-flow, and control-flow analyses, decision tables, design reviews, and code walk-throughs.

Other Technical Aspects

StP addresses changeability by annotating objects in the model and tying these objects to requirements and/or requirements documents, so that one can associate requirements with parts of the analysis and design models. Extensive checking programs are applied to the model to assist in the early detection of errors. The method is designed to ensure consistency within the system by enforcing consistent name use, and by means of a shared data dictionary for analysis and design information. There are tailorable formats for the data dictionary and design document, both of which are automatically generated from previous steps.

3.41.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

StP provides guidelines and automated support for assessing complexity and configuration management.

Communication Channels

Within the development team, the features of the method designed to facilitate communication are a multiuser relational DBMS, automatic documentation, version control, and locking. Between the development team and management, and between the client and the development organization, automatic documentation, data-flow diagrams, and entity-relationship diagrams are used to promote communication. Additional involvement of the client may happen if the user employs the User Software Engineering component, in which case the client works with prototypes of the user interface design.

Quality Assurance (Tables 12,14,15)

Testing activities are not addressed; however, StP incorporates programs designed for early detection of inconsistencies and errors in the model of the system.

Documentation Formats (Table 16)

All documents required to be produced are tailorable in format and generated automatically based on data produced from other steps in the method.

3.41.5 EASE OF USE

Technology Insertion

The developer estimated that, for successful use of the method, a development team leader should have as a minimum a bachelor's degree, three to five years of development experience, working knowledge of two programming languages, and experience on two different software systems.

Training assistance is offered in a variety of ways. There are hands-on demonstrations, overview presentations, classroom tutorials, and on-site consulting by the vendor and independent consultants. Also, assistance includes an on-line help facility, a "hot-line" service, user manuals, a users' support group, related publications from third-parties, and periodic technical updates.

The developer estimated that a project manager would need two days to acquire an understanding of the major features of the method. An experienced developer would also need two days to learn to use the method's essential features, and three months to achieve the level of expert user.

Automated Facilities

The method provides within itself automated support for several modes of representation, documentation required, and for associated project management activities. See Tables 9, 10, 13, and 16.

3.41.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

Configurations appropriate for hosting the method include DEC, Sun, Apollo, and HP workstations.

Acquisition Costs

Cost for a single user to acquire the method and required components ranges between $5-20 K. For multiple users and training costs, contact the vendor. The licensing policy is based on the number of concurrent users.

Contact Information

Interactive Development Environments, Inc.
595 Market Street, 12th Floor
San Francisco, CA 94105

415-543-0900

[provider]

3.41.7 REFERENCES

[Wass86] A. I. Wasserman, P. A. Pircher, D. T. Shewmake, and M. L. Kerster, "Developing Interactive Information Systems with User Software Engineering Methodology", IEEE Transactions on Software Engineering, Vol. SE-12, No. 2, February 1986, pp. 326-345.

[Wass87] A. I. Wasserman and P. A. Pircher, "A Graphical, Extensible Integrated Environment for Software Development", ACM SIGPLAN Notices, Vol. 22, No. 1, 1987, pp. 131-142.

[Wass88] A. I. Wasserman, P. A. Pircher, et. al., "A Generalized Annotation Mechanism for Objects in a CASE Environment", Proceedings of Software Engineering and its Applications, Toulouse, France, Dec. 1988.

[Wass89] A. I. Wasserman, P. A. Pircher, and R. J. Muller, "An Object-Oriented Structured Design Method for Code Generation", SIGSOFT Software Engineering Notes, Vol. 14, No. 1, Jan. 1989, pp. 32-55.

Further information about methods which have been incorporated into the StP environment can be found in [Chen76], [Gane79], [Hatl87], [Jack75], and [Your78].

3.42 STRADIS -- Structured Analysis, Design and Implementation of Information Systems

3.42.1 BACKGROUND

Synopsis

This method begins with the development of an Information Systems Transition Plan covering the organization's information systems requirements over the next three to seven years. The information system development process uses structured analysis to define the logical model of the system. A top-down decomposition strategy is used in the design phase, and the method emphasizes structured programming techniques and choosing meaningful names during the development phase. For major development projects, the developer considers STRADIS most useful in the areas of transaction processing systems, management information systems, and decision support systems.

The method is available for general use.

History

STRADIS was developed by C. Gane, P. Sarson and McDonnell Douglas personnel. The developer states that STRADIS "was the first method developed around the use of the tools and techniques of structured analysis and design." It was first used with respect to a deliverable system in 1980.

3.42.2 DESCRIPTION

Structured Analysis, Design and Implementation of Information Systems, STRADIS, recommends the development of an Information Systems Transition Plan to organize the information systems requirements over the next three to seven years. In addition to the Information Systems Model, a Business Information Model is developed along with a Hardware/Software/Network Plan. The software development process consists of five major phases: Analysis, Design, Development, Installation, and Maintenance.

STRADIS is founded upon several approaches to development, including data flow-oriented, data structure-oriented, entity-relationship modeling, and functional decomposition. Concepts which are essential to the method are stepwise refinement, process abstraction, structured programming, and module coupling/cohesion. Guidelines are provided by various project types and sizes to skip and/or combine workproducts over the life cycle. This method is well-suited to the Waterfall model, although effective use of STRADIS is not dependent upon any particular paradigm of the software process.

The analysis phase starts with a request for a data processing capability. The response to this request is an initial study to assess benefits from either building a new information system or changing an existing one, and to estimate the cost and time to do a more detailed study. The detailed study constructs a logical model of the current system, draws up the objectives which the new system would have to meet, and constructs a logical model of the new system which incorporates these objectives. A second estimate is provided for the cost and time of developing and operating such a system.

The Draft Requirements Statement is the document which links the application analyst with the expert technical systems designers. This document refines the new system's logical models and objectives, and adds the constraints for any physical design. The intent of the document is to establish what the new system is required to do so that cost-effective strategies for its implementation may be created. Accordingly, one or more Outline Physical Designs are produced to explore different technical solutions. This physical design component is used to refine the Draft Requirements Statement into a Total Requirements Statement. This is done by developing explosions, or expansions, of constructs, defining the logic of the basic application processes with structured English, and completing the system data models.

The design phase is composed of three intertwining activities: computer process design, physical data design, and human interface design. This phase concludes with a Design Statement.

The development phase requires that each activity be repeated for each version of the system to ensure the correctness of interfaces.

The installation phase involves installation of hardware, training, resolving any software conflicts with pre-existing systems, tuning for response, converting files and testing the system. This phase may be done in parallel with the old system. Additionally, a Post-Installation Audit Report is prescribed after users have had a chance to evaluate the new system. It is recommended that this report be repeated every two years.

Maintenance involves responding to the requests for changes and enhancements in a prioritized way, and communicating with the users about new releases or urgent fixes.

3.42.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

STRADIS was judged by the developer to be particularly well-suited for applications involving distributed processing or networks, data processing and databases. The developer reported that the method has been used successfully on a wide variety of business applications for many industries, including aerospace, banking, city and state government, insurance, military, manufacturing, securities, telephone company, and utilities.

The method is intended to be used on projects of all sizes. It has been used to develop small, medium and large projects, totalling more than 250 delivered systems from over 100 user organizations. Implementation languages most frequently used for coding systems developed with STRADIS are COBOL, Ada, C, and FORTRAN.

Target Constraints

The developer provides a tool, ProKit*Workbench, which evaluates timing constraints for both processes and data flows by capturing the elapsed time for a given process or transaction.

Special features of the target hardware architecture can be addressed by strategic design procedures, which use knowledge about a particular physical environment to allocate a set of logical requirements (design units) to that environment.

Modes of Expression (Tables 9,10)

The method requires specified documentation templates, narrative overviews of modules, and structured English as its textual representations. Required iconographical representations are data-flow and entity-relationship diagrams. Depending on the application, the method strongly encourages several other forms of representation.

Translating from one mode of expression to another is accomplished by rules prescribed for mapping data flow diagrams into structure charts. Regarding transformation across the phases of the software process, the developer stated that workproducts flow smoothly through the development life cycle and traceability is provided from start to finish. Use of the tool, ProKit*Workbench, assists in migrating known facts from analysis to design via linkages established by the designers.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

In the context of prototyping, dynamic animation is strongly encouraged by the method. Data-structure and data-flow analyses are both required, and control-flow analysis is encouraged as part of data-flow analysis.

Other Technical Aspects

The developer stated that requirements are developed or defined in a modular fashion, resulting in an independent set of components among which the impact of change is clear. It was felt that the method fosters one fact in one place, thereby facilitating the incorporation of changes via an automated tool environment.

Change is made in the highest level of documentation in which it would be visible and then migrated through the remaining layers. The method also advocates the use of a central data dictionary for maintaining consistency within the developing software. Early detection of inconsistencies or errors is addressed by means of walkthroughs and user participation on the development team. Additionally, ProKit*Workbench can be used for detecting rule violations.

Assistance in identification of possible reusable components is provided by defining specific, single-function, independent design units during design and modules during coding.

3.42.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

STRADIS has a number of procedures for conducting activities associated with project management. The method recommends allocation of people to different activities. The method recommends the writing of a General Project Plan to establish milestones for the project, which is reviewed a number of times and revised upon significant changes occurring that impact cost and/or completion date. There is also provision for detailed estimation of the man-time and elapsed time required to accomplish each milestone.

Communication Channels

Communication within the development team is facilitated by the use of walkthroughs, technical reviews, and a project repository (dictionary). Management reviews that oversee the entire development effort and management authorizations for funding are intended to facilitate communication between the development team and management.

Between the development organization and client, the method provides walkthroughs and roles that define the type and level of user participation and responsibility on the project team. The users are members of the development team with specific roles to play during the entire development life cycle. Also provided are prototyping and extensive use of graphic models for process, data and behavior (man/machine interaction).

Quality Assurance (Tables 12,14,15)

STRADIS provides specific directions for test planning at one or more precise points in the method, unit or integration testing, field testing, generation of test data, and regression testing. A required document of the method is the quality assurance/test plan. Review techniques utilized are design reviews, code walk-throughs, and Change Control Board reviews.

The method also provides directions for recording specification or design options, trade-off studies, rationales for decisions, personnel involved in making decisions, and changes related to specification or design decisions.

Documentation Formats (Table 16)

A number of documents are required to be produced when using STRADIS. Their tailorability varies and some of them are available via ProKit*Workbench.

3.42.5 EASE OF USE

Technology Insertion

The developer estimated that a development team leader would need less than two years college-level technical education, one to two years of development experience, knowledge of one programming language and experience with one software system in order to successfully use the method. The major constructs necessary for understanding by an experienced developer are data flow diagrams, data models, menu navigation diagrams (state transition), decision tables, and structure charts.

Available training includes overview presentations, classroom tutorials, on-site consulting by the vendor, a "hot line" service, user manuals, a users' support group, and periodic technical updates.

The estimate of learning time for a project manager to acquire an understanding of the major features and benefits of STRADIS was five days. Estimate of time for an experienced developer to learn the essentials was ten days, and six months to become an expert user. The developer indicated that a gauge of expert user level could also be the number of completed projects using the method which are needed to fully understand the method.

Automated Facilities

The developer provides ProKit*Workbench, a tool which provides support for the following: data flow diagrams, entity-relationship diagrams, process narrative capture, completeness/consistency checking, structure chart creation, prototyping, data dictionary population, and selected deliverable component generation.

3.42.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

An IBM PC-XT (or strict compatible) and higher PC configurations are appropriate for hosting tools associated with the method.

Acquisition Costs

Per company, the cost to acquire the method and required components is $48,000. Unit cost for ProKit*Workbench is $9,200 for a single user, and $6,500 for 10 or more users. Technical training costs $190 per student per day, and $2,500 per class per day. A management overview is $2,500 per class per day.

Contact Information

McDonnell Douglas
P. O. Box 516, Dept. L860
St. Louis, MO 63166-0516

314-232-8997
800-325-1087
[Provider and developer]

3.42.7 REFERENCES

[Gane79] C. Gane and T. Sarson, Structured Systems Analysis: Tools and Techniques. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1979.

3.43 TAGS -- Technology for the Automated Generation of Systems

3.43.1 BACKGROUND

Synopsis

The TAGS system is composed of three elements: an iconographic input/output requirements language, computer-based tools supporting design of systems or software, and a development method based upon stepwise refinement. The requirements language records system design concepts emphasizing traceability between requirements and implementation and may be used both in developing new systems as well as in analyzing existing systems. The associated tools support several aspects of the software process including validation and verification, configuration management, documentation generation, and the simulation of design behavior.

History

TAGS was developed at Teledyne Brown Engineering principally by G. J. Gotvald and R. E. Alger. An early version was first used with regard to a deliverable system in 1976. Pilot projects were undertaken in 1983 [Siev85].

3.43.2 DESCRIPTION

The Technology for the Automated Generation of Systems (TAGS) is a systems/software engineering method addressing the creation of real-time embedded computer systems for the DoD. It consists of a graphical, executable specification language, a supporting work station and software tools. The entry point into the software process occurs during the development of an early specification. Using the Input/Output Requirements Language (IORL) and the TAGS work station, icons representing the high-level components of a system can be assembled and interfaced with others representing the environment with which the system functions. These diagrams emphasize the identification of independent system components, their interfaces and data flow rather than control at this level. The system to be developed is elaborated through stepwise refinement (functional decomposition). Predefined processes or stubs and control-flow semantics are incorporated into logic diagrams as the elaboration continues.

The system model assembled in the above description is treated as a prototype. The software tools supporting the TAGS system can be used to make correctness checks of the prototype in both syntactic and semantic ways. A static tool may be used to check interfaces and proper use of the IORL language. In addition, a dynamic tool may be used to simulate the behavior of the target system by exercising the prototype. A configuration management tool is also part of the support systems, as well as a natural language processing tool which catalogs requirements for traceability to the IORL design database. Finally, an automatic code generator can be used to produce Ada code directly from the IORL model.

The repeated execution of the development steps implies a method which is evolutionary, or prototyping iteratively, applied to the early part of the software process. The developer states that proceeding through the steps several times, making revisions as needed, offers an opportunity to better understand the system being designed and/or maintained.

3.43.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

The developer considered the method well-suited for applications involving embedded systems or process control, time-critical or real-time processing, scientific or engineering, distributed processing or networks, and large scale simulation or modeling. Examples of delivered systems developed with the method include Trident Submarine program updates, BM/C^3 Control Centers for Ballistic Missile Defense experiments, E3 AWACS upgrade program, and medical instrumentation systems.

TAGS has been used in between 21 to 50 organizations, yielding between 21 to 100 delivered systems. The method is intended for use on medium and large projects; it has been used on small and medium ones. Implementation languages most frequently used when coding systems developed with TAGS are FORTRAN, C, and Ada.

Target Constraints

The method addresses timing constraints on the target system by embedded I/O definitions in each system component's control flow. Schematic Block Diagrams identify process parallelism, while controlled AND and OR constructs identify task concurrency. Probability of failure can be specified on communications interfaces.

Portability is assisted in the sense that TAGS formulates an implementation-independent design. Through the generation of Ada, cross-compiling can be done for different target configurations.

Modes of Expression (Tables 9,10)

Required textual modes of representation include a formal specification language and mathematical notation. Specified documentation templates are strongly encouraged. TAGS provides automated support for all of the above. Data-flow diagrams and control-flow diagrams are required iconographical modes of representation.

The method has several rules for mapping one mode of representation to another. In particular, Schematic Block Diagrams can be translated into a hierarchical physical decomposition. Input/Output relationships are mapped into hierarchical functional decomposition training diagrams, and predefined process diagrams go to recursion/reuseability.

Transformation across phases of the software process is facilitated by decomposing and elaborating the system concept until the design implementation level is reached. All system functions are considered mathematical transformations with inputs and outputs. At the design level, these mathematical constructs are automatically inserted into Ada source code templates for either simulation or target environments.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

Essential to TAGS for clarifying system requirements or behavior are rapid prototyping, simulation, and executable specifications. Required analysis and review techniques are data-flow analysis, control-flow analysis, Facilitated Application Specificaiton Techniques, and Change Control Board reviews.

Other Technical Aspects

The method assists in incorporating changes in the requirements by on-line requirements editing and analysis of subsequent ripple effects of the change. Consistency is maintained among specification, design, or code by means of consistency checks with static and dynamic analysis tools. Ada code is automatically generated from the design.

The TAGS Predefined Process Diagrams can be placed into reuseable libraries which are at a higher level of design abstraction. Schematic Block Diagram and Predefined Process Diagram trees identify recurrent logic areas.

3.43.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

TAGS has specific procedures for addressing configuration management; this activity is provided with automated support. Automated support also exists for tracking project progress. Estimating initial cost, providing incremental data about expenditures, and tracking project progress are all addressed in terms of guidelines for accomplishing these activities.

Communication Channels

Facilitation of communication within the development team is accomplished in several ways. First, interfaces are explicitly defined in only one location between system components. Moreover, there is one universal unambiguous system design language which serves as a foundation for clear communication. Thirdly, the actual design can be used for design walk-throughs and reviews. Finally, TAGS incorporates automated configuration management of changes or revisions.

Between management and the technical development team, TAGS coordinates communication by its visual indication of design, by supporting a quantitative measure of design progress, by using simulation results to exhibit design progress, and by identification, via configuration management, of changes and their rationales.

The graphical design is the foundation for any design walk-throughs or design reviews with the client. Also, the client can audit the database at any time, and explicitly define the system requirements using the method.

Quality Assurance (Tables 12,14,15)

A framework is provided by TAGS for addressing testing activities; automated support is provided for prescriptive checking of interfaces.

Automated recording procedures are provided for maintaining a record of specification or design options which were considered, any trade-off studies, and all changes related to specification or design decisions. The method provides specific directions for maintaining a record of the rationale for any decision and the personnel who were involved in making a decision. Configuration management is specifically addressed at an automated support level.

Documentation Formats (Table 16)

The documents required to be produced are fixed in format, with the exception of internal program documentation, which is tailorable in format. Automated support is provided for these documents. For specific information, see Table 16.

3.43.5 EASE OF USE

Technology Insertion

Minimum qualifications for a development team leader's successful use of TAGS were given as a bachelor's degree, one to two years of development experience on one software system, and knowledge of one programming language. In addition, control theory should be understood.

Training assistance is available in the form of hands-on demonstrations, overview presentations, classroom tutorials, user manuals, "hot-line" help facilities, on-site consulting, and video tapes.

An overview presentation for management requires one day. An experienced developer should be able to learn the essentials in five days. One and a half months would be needed for an experienced developer to reach the level of expert TAGS user.

Automated Facilities

The tools included within the TAGS system are: an on-line storage and retrieval system, a "Diagnostic Analyzer", a "Simulator", a configuration management support tool, and a series of document processors. The diagnostic analyzer supports checks for correctness within the structure of the design while the simulation tool supports checks of the behavior of the design. Other computer-based support tools provide an on-line storage and retrieval capability. Finally, computer source code can be generated directly from the executable specification. See Tables 9, 13, 14, 15, and 16.

The developer referenced two tools from other vendors which were felt to support documentation publishing activities associated with TAGS. They were Engineering Writer from Context Corporation, and Technical Publishing (TAS) from Interleaf.

3.43.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

Configurations appropriate for hosting the tools associated with TAGS are Sun/OS 3.5, Apollo/Aegis 9.7, DEC VAX Station 2000 (Ultrix 2.2), and IBM PC RT/AIX (in progress).

Acquisition Costs

The unit cost to acquire the method and required components is $7,500. Cost for other recommended components supplied by the method vendor is $10,500. Technical training is $1,500, and there is no charge for a management overview.

The licensing policy is per workstation node discount schedule. There is also a site license discount.

Contact Information

Gerard J. Gotvald
Teledyne Brown Engineering
Cummings Research Park
Huntsville, Alabama 35807

205-532-1613
800-633-4675

[Provider]

3.43.7 REFERENCES

[Siev85] G. E. Sievert and T. A. Mizell, "Specification-Based Software Engineering with TAGS", IEEE Computer, Vol. 18, No. 4, April 1985, pp. 56-65.

3.44 UCRA -- User-Centered Requirements Analysis

3.44.1 BACKGROUND

Synopsis

User-Centered Requirements Analysis is a method for developing detailed requirements specifications for a software application. It modifies and combines other process and data modeling techniques in a manner designed to focus on communication between users and developers. It provides users with the opportunity to make key decisions, while providing the developers with a detailed understanding of the application requirements. The method is effectively in the public domain, based upon the use of the associated text.

History

Charles Martin developed User-Centered Requirements Analysis while managing systems analysts in consulting firms from 1980-85. The method combines concepts from Yourdon/DeMarco and Gane/Sarson process modeling based on data flow diagrams, with Bachman, Chen, and James Martin data modeling techniques using entity-relationship diagrams, and with requirements verifiability from military standards. The method was used and refined on twelve system definition projects from 1984-85. A book detailing the method was published in 1988.

3.44.2 DESCRIPTION

User-Centered Requirements Analysis (UCRA) employs a stepwise refinement process to define requirements for a software application in a manner intended to be in terms end-users can understand and critique. Rather than a vague, user-defined requirements specification, followed by a developer-defined system specification, a single detailed requirements specification is developed with heavy user involvement. The resulting specification is "owned" by the user and intended to be sufficiently complete for developers to skip the traditional system specification step.

The method starts with diagrams and then provides verifiable requirements detail. Treating processes and data as equally fundamental, UCRA models functional processes and data bases in parallel. "User concept diagrams" add icons and user-interaction notations to traditional data flow diagrams. "Canonical form data structure diagrams", a type of entity-relationship diagram, show data entities and relationships in a manner designed for users to understand. The diagrams are supplemented with textual detail.

UCRA prescribes a series of thirteen mini-steps for generating the requirements specification. The method covers many of the concerns in creating a requirements specification, including coordination of numbering systems, melding of diagrams and text, performance requirements, and assessment of the requirements in order to determine risk and feasibility.

3.44.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

UCRA is designed for applications with extensive user interaction and an integrated database. Thus, it is best-suited for data processing or database applications. It can also be used to support the parts of scientific, engineering, or real-time applications which deal with a database and user interaction. For example, in a command and control system, it could help describe the subsystems which display status to users and accept human decisions. It is best-suited for medium to large projects. It may also be used on small projects when there are several user types or relatively complex logical connections between the functions and data.

The developer estimates that more than one hundred organizations have used this method in developing between one hundred to two hundred fifty delivered systems, including financial planning systems, procurement tracking systems, executive information systems, command and control systems, an expert system configurator, and a manufacturing planning system. The implementation languages most frequently associated with the method are fourth-generation database management system (DBMS) languages, C, Fortran, and COBOL.

Target Constraints

The method focuses on defining requirements for target systems involving a large amount of user interaction and an integrated database. The method developer states that the method prescribes steps for handling frequency and completion times of major processes, as well as response times and security of access. Concurrency issues are addressed indirectly by specifying the workload requirements. The method gives no specific guidelines for handling fault-tolerance issues, but allows these to be defined in a textual section customized for the application.

Since design techniques are better tailored to the target environment, people using UCRA will need to tailor their own transition from requirements to the design technique they select for their environment. If the operating system or the DBMS is pre-selected, or if other special features of the target hardware architecture are known with requirements, these are addressed as constraints on the design. If portability among various target environments is an issue, that issue is also addressed with environment design constraints. Concurrency issues are addressed only in an indirect manner by specifying workload requirements.

Modes of Expression (Tables 9,10)

The method requires narrative overviews of modules. It suggests a "natural" specification outline, but also accommodates prescribed formats such as Mil-Std 2167A. It also encourages the use of structured English and decision tables. The diagramming basis for representing requirements is composed of the user concept diagrams, the data structure diagrams, and hierarchy charts.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

UCRA provides a general process for requirements generation. It requires data-structure analysis, and encourages data-flow analysis and the use of decision tables. Although the method uses data-flow diagrams, and hence identifies data flows, it concentrates data analysis on data stores. The method developer claims this avoids "overworked" data flows. The method recommends deciding, for each step, whether individual client interviews or group sessions are appropriate. The method encourages the use of incremental or evolutionary development to clarify the expected behavior of the system with the client.

Other Technical Aspects

The developer states that the method assists in reducing the effort needed to fully incorporate changes in the requirements and assists in the early detection of inconsistencies. The structure of the requirements specification simplifies locating the requirements to be changed and the process prescribed by the method weeds out requirements redundancy. The diagrams utilized are designed to be effective at pointing out inconsistencies early in the requirements analysis.

A requirements set may be used as a starting template for requirements of a similar application system. At a lower level, verifiable functional requirements for standard capabilities, such as an hoc query, can be reused as a "boilerplate" in another requirements specification.

It is expected and recommended that system design be highly tailored to the specific development environment to be used. The transition from UCRA requirements to design must be worked out for each target environment.

3.44.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

UCRA prescribes specific directions and procedures for analyzing risk, project planning, and scheduling. It provides guidelines or a framework for estimating initial cost, projecting cost of completion, and other project management activities.

The developer states that the standardized diagrams and textual output promoted by this method provide a consistent basis for review of specifications by senior analysts, project leaders, and high level managers and that from this start it is much easier to manage the rest of development and implementation.

Communication Channels

UCRA focuses on communication between the client and the development organization. Detailed user requirements are developed in conjunction with end-users. These requirements are detailed in the terminology of the user community, and stress diagrams and prose techniques which are understandable to the end-user.

The top-down approach is intended to highlight the business objectives of the system for managers, and let the managers ensure that these are correct in order to guide the systems analysts, user representatives, and developers. Elaborating functional requirements to a verifiable level and data requirements to the data element level provides developers with the detail they need to carry out their designs. The resulting model permits checking for consistency and operability.

Quality Assurance (Tables 12,14,15)

UCRA places emphasis on the requirements analysis process and clarification of user expectations. Special process and data diagrams are used in conjunction with a top-down approach addressing functions and data simultaneously to aid in the early detection of inconsistencies and/or errors. Providing a consistent basis for review of specifications, UCRA facilitates design reviews, JAD-like approaches, and Change Control Board review. This also serves to isolate specific functional and data requirements for reference in acceptance testing.

Documentation Formats (Table 16)

UCRA provides detailed guidance in preparing requirements definitions and functional specifications. While the method is independent of documentation requirements, it can support customized templates and it has been used to provide the functional description and data requirements documents required by DoD-Std 7935, Automated Data Processing Systems Documentation, as well as the documentation formats prescribed by Mil-Std 2167A.

3.44.5 EASE OF USE

Technology Insertion

The basics of the method are described in Charles Martin's book and a professional development seminar is available from Charles F. Martin Associates and Digital Consulting, Inc. The seminar addresses issues such as verifiable functional requirements, completeness, prototyping, benchmarking, and risk reduction strategies. On-site consulting is also available.

The developer stated that a project manager would require one day to acquire an understanding of the major features and benefits of the method, while an experienced developer would require three days to learn the essentials. An experienced developer would need to go through a total requirements specification effort using the method, typically requiring three months, to achieve the level of expert user of the method.

The minimum qualifications for a development team leader in order to use the method include a bachelor's degree and three to five years experience with three to four different software systems. An experienced developer should understand entity-relationship modeling and have the ability to write clear English for successful use of the method.

Automated Facilities

The method can be incorporated into an integrated environment supporting other methods for later stages of the development process. Automated tools which support activities of this method include the following:

Name	Tool Vendor	Activities Supported
Excelerator/Customizer	InTech	UCRA
Design Aid	Nastec	UCRA
Cadware Toolkit	Cadware	UCRA
Design	Meta Software	UCRA
Data Analyst	Bachman	Data Requirements
Auto-Mate Plus	LBMS (& Cullinet)	Data Requirements

Other customizable CASE tools, DBMS packages, or interactive graphics systems may also be used.

3.44.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

Ordinarily a database management system will be used, but it is not required. The hardware requirements would depend on the automated tools selected.

Acquisition Costs

The only required cost for acquisition is $40.00 for the textbook. The three day seminar costs $1,000 per person with a reduction to $250 for high volume enrollments. The cost of consulting help on a trial project is $10,000 to consult with one person, or $5,000 per user with a volume reduction to $1,000.

Contact Information

Note: UCRA is not a standard product name.

Charles F. Martin Associates 617-371-7011
P. O. Box 91
Concord, MA 01742 [Developer]

Digital Consulting Associates 617-470-3880
6 Windsor Street
Andover, MA 01810 [Training]

3.44.7 REFERENCES

[Mart88] Charles F. Martin, User-Centered Requirements Analysis. Englewood Cliffs, NJ: Prentice-Hall, 1988.

3.45 WARD/MELLOR -- Ward/Mellor Real-Time Method

3.45.1 BACKGROUND

Synopsis

The Ward/Mellor Real-Time Method provides an extension of structured analysis and structured design techniques for use in developing real-time applications. It addresses the software development activities of specification, preliminary design, and detailed design, and provides simulation and prototyping capabilities through execution of the data, control, and timing information incorporated in the iconographical model.

History

Developed by Paul T. Ward and Stephen J. Mellor, this method provides one of several real-time extensions to Yourdon's SA/SD. Ward/Mellor is primarily based upon DeMarco's Structured Analysis (SA) and Yourdon/Constantine's Structured Design (SD), which are the components of SA/SD, and McMenamin and Palmer's, Essential System Analysis, a 1984 modification of SA/SD. The method is also based upon Chen's entity-relationship diagramming.

Ward/Mellor's technique "transformation schema" extends the data flow diagram as described by DeMarco for SA, by Gane and Sarson for Stradis, and by Ross and Brackett for SADT. The idea for the method's dynamic execution capability is based loosely on a version of Petri net execution. This method was first used for a deliverable system in 1982.

3.45.2 DESCRIPTION

The Ward/Mellor Real-Time Method (Ward/Mellor) first emphasizes understanding the world in which the system is to operate. This is viewed in terms of events to which the system must respond and in terms of an information model, describing objects, relationships between them, and their attributes. The method utilizes an extension of the data flow diagram, called the "transformation schema", to build a comprehensive system model representing the data, control, and timing aspects of systems. The transformation schema is heavily dependent on the objects defined in the information model and the results of its execution provide a behavior pattern model.

The transformation schema permits the creation and evaluation of two different types of system models: the Essential Model and the Implementation Model, as in ESA [McMe84]. The Essential Model shows the system as a virtual machine with infinite resources, while the Implementation Model assumes a real machine with constrained resources. The method uses multiple simultaneous views to model a system in terms of data transformations and control transformations. To reduce the complexity of modeling a large system, Ward/Mellor applies a hierarchical presentation scheme to both transformations and flows, creating a multi-leveled set of schemas. Most of the modeling activities are carried out concurrently with continuous interaction among activities during successive refinement.

The method provides simulation and prototyping capabilities through transformation schema execution. Execution rules utilize tokens to indicate actual or potential activity of some element depicted in the schema notation, such as transformations, flows, and buffers. The types of actions to take place depend upon the type of element on which the token is placed. Execution rules can be incorporated into an execution plan in which a table is used to show a series of interactions each occurring at a point in time.

To differentiate it from Hatley's real-time extensions to SA/SD, the developer states that this method emphasizes sequential rather than combinational logic and provides guidelines for event-entity/relationship based model-building rather than top-down model-building.

3.45.3 TECHNICAL ASPECTS

Applicability and Usage (Tables 4,5,6)

Ward/Mellor is intended for use in developing real-time systems and embedded systems, as well as for process control, systems programming and distributed processing. It has been used to develop systems for industrial process control, avionics, screen-based man-machine interfaces, weapons platforms, and embedded micro-based controllers for consumer electronics items, photocopiers, and logic analyzers. It has been used by more than 100 organizations in developing more than 250 medium and large systems in FORTRAN, C, Assembler, and Ada.

Target Constraints

The model is inherently concurrent and was designed to focus on such issues. Timing constraints may be attached to the specification model. Special features of the target operating system can be modeled using the same notation as for application system specification and design. Fault-tolerance issues and security of access can be modeled within the design model.

Modes of Expression (Tables 9,10)

Ward/Mellor uses multiple simultaneous views to model a system. The basic notation consists of data transformations, which are similar to data-flow diagrams, and control transformations. Control transformations, represented by dotted circles, control the behavior of the data transformations through activation or deactivation. Continuous and discrete data flows are signified as well as three types of event flows, all of which are discrete messages with no variable content. The event flows are differentiated to express the different states of knowledge of the sender. Data stores and buffers are iconographically represented, as well as finite-state information in tabular or diagrammatic format.

The method prescribes mapping rules for translating from entities with attributes and relationships with attributes to the process model data flows and data stores. Rules are provided for translating from events (stimulus in the system environment) to the Transformation Schema (response to the stimulus) and from the Transformation Schema to a structure chart.

Techniques for Analysis and Requirements Clarification (Tables 11,12)

In addition to requiring the use of executable specifications, the method encourages the use of dynamic animation, simulation, and incremental development to clarify system requirements and behavior. Refinement of the software design is accomplished using the method's provisions for data structure analysis, control flow analysis and simulation via execution of the transformation schema. Heuristics for splitting or merging finite-state machines and data flow models facilitate the transition between specification and design. The use of design reviews and decision tables is also encouraged.

Other Technical Aspects

The correspondence between specific portions of the specification model, (e.g., event responses and objects/relationships/attributes) and portions of the application domain facilitates identification of the portion of the model affected by a requirements change. The tracing tables recommended by the method tie together specification and design components. These can be used to track changes and ensure consistency among specification, design and code.

3.45.4 PROJECT CONTROL AND COMMUNICATION

Project Management (Table 13)

Project management and configuration management are beyond the scope of the method.

Communication Channels

The graphic notations, executable models, and guidelines for mapping features of the application domain into specific features of the model are designed to facilitate and coordinate communication within the development team. The graphic notations and the hierarchical organization of the final model are designed for communication between management and the technical development team, as well as between the client and the development organization.

Quality Assurance (Tables 12,14,15)

Although Ward/Mellor does not directly address testing, it provides a framework for the generation of tests based on system requirements. The method incorporates features within itself leading to verification, such as provisions for data-flow and control-flow analyses of dependencies, as well as execution of models. The detailed formation rules for model components are intended to provide rigorous consistency checks and to assist in early detection of inconsistencies and/or errors. CASE tool implementations automate these checks. Use of the tables recommended by the method provides a correspondence between the specification components and design components. The use of design reviews is encouraged.

Documentation Formats (Table 16)

The diagrams and structure charts required by this method provide documentation of the system requirements, as well as the high level and detailed design. The method does not explicitly prescribe external documentation, however model segments may be included in the deliverable documentation.

3.45.5 EASE OF USE

Technology Insertion

The developer recommends that minimum qualifications for a development team leader using this method include a Bachelor's degree, 3-5 years of development experience, a working knowledge of one programming language, and previous experience with two different software systems. For successful use of the method, an experienced developer would need an understanding of finite state machines, the data-flow model of computation, and the relational view of data. The estimated training time for an experienced developer to learn to use the method would be 10 days, and 6 months or more to achieve the level of expert user. A project manager could acquire an understanding of the major features and benefits of the method in 2 days.

Training is available in the form of hands-on demonstrations, overview presentations, classroom tutorials, and on-site consulting, while CASE tool implementations typically have on-line help, a hot line service, user manuals and users' support groups.

Automated Facilities

Use of the method is simplified through the incorporation of its modes of representation in many of the CASE tools currently available and under development. Such tools may be used to provide automated support for the graphic techniques of the method and to provide automated generation of documentation. Some are capable of performing consistency checks within and between the graphic representations. The available CASE products include:

Name of Tool	Tool Vendor	Activities Supported
Excelerator/RTS	Intech	Model Capture/Analysis
DesignAid 4.0	Nastec	"
Auto Asyst	Asyst	"
VS Designer	Visual Systems	"
Yourdon Analyst/ Designer Toolkit	Yourdon	"
Mentor Graphics CASE	Mentor Graphics	"
Autocode	Integrated Systems	ModelCapture/Analysis /Code Generation
SDAR	Athena Systems	Model Capture/Analysis/ Execution

For more information on automated support associated with this method, refer to Tables 9 and 10.

3.45.6 ACQUISITION FACTORS

Hardware/Software Configuration Required

The method has been incorporated in CASE tools running on IBM PCs and engineering workstations. It does not have any specific hardware or software configuration requirements for use in a non-automated mode.

Acquisition Costs

Unit costs to acquire the method and required components, including "typical" CASE tool software, were given as $7,000 for a single user, $6,000 for a low-volume user, and $4,000 for a high volume user. Technical training per developer is $2,500 for a single person, $2,000 for low-volume users, and $1,000 for high-volume users. Cost for a management overview is, per manager, $500 for a single person, $400 for low-volume users, and $100 for high-volume users.

The vendor will license training materials. The policy on CASE tool implementations varies by vendor.

WARD/MELLOR

Contact Information

(Note: The acroynm Ward/Mellor is not a standard product name.)

Paul T. Ward Software Development Concepts 424 West End Avenue 11E New York, New York 10024	212-362-1391 [Developer]

3.45.7 REFERENCES

[Ward86] P. T. Ward, "The Transformation Schema: An Extension of the Data Flow Diagram to Represent Control and Timing", IEEE Transactions on Software Engineering, Vol. SE-12, No. 2, Feb. 1986.

[Ward85] P. T. Ward and S. J. Mellor, Structured Development for Real-Time Systems. New York: Yourdon Press, 1985.

[Ward89] P. T. Ward, "How to Integrate Object-Oriented Design with Structured Analysis and Design", IEEE Software, Vol. 6, No. 2, March 1989, pp. 74-82.

(See also [DeMa78], [Gane79], [McMe84], [Pete77], [Ross76].)

This page is intentionally blank.

CHAPTER 4

EMERGING METHODS

This chapter contains descriptions of methods which were classified by the developer to be in "development", "beta test", or "exploratory research", as opposed to being in general use. There is, however, great variation in the degree to which these methods have progressed in their evolution. Some methods have been used to develop software on pilot projects, others are being used in beta sites for refinement purposes, and still others are variations of existing methods whose basic tenets are known in published literature. Finally, there are descriptions for methods that are experimental and untried.

The descriptions of methods in this chapter follow the same sequence in presenting information as that used in the previous chapter; however, the amount of information has been abbreviated, and some information has been omitted when it was inapplicable to emerging methods.

The reader should note that methods included in this chapter are in a state of flux. A method which is currently immature may very well come into use at some point in the future; another method, encapsulating experimental ideas, may exert an influence on the software process, but itself disappear. Accordingly, the information presented in this chapter represents a small sampling of possibilities for the future.

ABDP

4.1 **ABDP** -- Actor Based Design and Prototyping

4.1.1 BACKGROUND

More a design philosophy than a method thus far, Actor Based Design and Prototyping is concerned with the development of guidelines for designing concurrent object-oriented systems which incorporate the notions of inheritance, dynamic binding, and client-server message-passing. Developed by D. A. Thomas and B. M. Barry, it is loosely based on the actor model of programming developed by Hewitt, Lieberman, Agha and others. Its approach to process structuring is derived from the Smalltalk model. Although any object-based language with support for concurrency is a candidate implementation language, the current realization utilizes Smalltalk and C. The method is in the exploratory research stage and has been used by the Defence Research Establishment Ottawa in developing an Electronic Support Measures (ESM) testbed for prototyping new signal processors.

4.1.2 DESCRIPTION

Actor Based Design and Prototyping (ABDP) is concerned principally with exploratory programming projects in which requirements tend to be vaguely stated at first, and must be evolved as the project progresses. Initial requirements are used as the basis for a simulation or animation of the required software system and its real-world environment. Details can then be supplied, as required, to expand this simulation by stages into a solution.

Providing a framework for system specification, design, implementation, and some project management activities, ABDP is founded upon an object-oriented approach and compatible with data flow-oriented and event-oriented approaches, entity-relationship and binary-relationship modeling. Based upon prototyping and evolutionary development, it is also well-suited for use in conjunction with Boehm's spiral and compatible with operational and 4GL models.

The major characteristic of this design approach is that the entire system, hardware and software, is described by personification of "actors". The actors in ABDP are not nearly as fine-grained as in Hewitt's actor model, rather they may be thought of as a group of cooperating objects which follow a single thread of execution. Each actor can then be considered to be a task, with the non-actor objects it encapsulates accessible by the computations that are carried out by that task. In particular, ABDP will refer to individual actors as clients, couriers, administrators, servers, workers, etc., based on the personified role played by each.

Actors synchronize their activities and communicate by sending one another messages. Actors have local state variables, but there is no global state in the usual sense. They are organized hierarchically, and use delegation to share responsibility for performing tasks. This provides a mechanism for controlling inter-task communication without imposing undue restrictions on the system designer. Each actor will typically have a supervisor and a number of subordinates. Actors which share a common supervisor are referred to as colleagues. By default, an actor is assumed to know (and hence to communicate directly with) only its supervisor, colleagues, and subordinates. If a designer wishes to have direct communication outside this default scoping, it must be provided for explicitly.

With ABDP, the first steps are to identify the actors by their behavior or function as follows:

- Decide on the natural kinds of actors to have in the system and to model the objects in the system's environment,
- Decide what messages each actor is to receive,
- Decide what actions each kind of actor should perform when it receives each message.

The results of this first step in the design process may be expressed by message send graphs for each actor which document communication with its supervisor, colleagues, and subordinates. Inheritance trees and parts hierarchy diagrams are also useful at this stage.

This initial simulation is then evolved through a series of increasingly detailed prototypes, until a final systei.ı solution is reached. Each prototype is evaluated against performance goals, deficiencies are identified and documented, and a plan for changing or augmenting the prototype is developed. When functional performance goals have been met, real-time performance is measured and time-critical sections of code are optimized.

Related efforts are in progress developing Orwell, a configuration management system designed to support team programming in large object-oriented applications, and researching Smalltalk instrumentation and performance measurement tools.

4.1.3 TECHNICAL ASPECTS

Intended to address multiprocessing, multitasking object-oriented systems, the method is well-suited for embedded systems, process control, distributed processing, and large scale simulation. The respondent states that it is compatible for use in other areas including real-time, scientific/engineering, systems programming, data processing, expert systems/artificial intelligence, and image processing/pattern recognition.

An electronic warfare simulation environment has been developed to support both the design and implementation phases of the ESM project. Simulations provide needed feedback to evaluate various versions of the system design, develop processing strategies, test algorithms, identify causes of poor response and evaluate the cost-effectiveness of proposed modifications. Based on the Smalltalk programming language, the ESM simulation environment supplies reusable objects which can be used to model the basic subcomponents of ESM systems and essential features of the environment, as well as user interfaces.

[Barr87] describes this approach for organizing collections of cooperating tasks in the multiprocessor. It uses Harmony, a portable real-time multiprocessing kernel that supports both multiprocessing and multitasking with priorities. With this kernel, any task can send a message to any other task in the same or a different processor and tasks can be moved to other processors without modifying the application software. Within the ESM testbed, two object-oriented languages are used concurren'ly, Actra and Objective-C. ABDP has extended Smalltalk to support the actor concept, resulting in Actra. Objecuve-C [CoxB86] is an object-oriented dialect of the C language.

This hybrid approach was devised for ESM to offer the modularity and robust response needed in light of the rapid, continual evolution of technology, techniques, and system requirements. All real-world objects are modeled by software objects, hence changes in requirements can usually be expressed naturally in terms of existing components. The object-oriented paradigm lends itself to weakly coupled modules which isolate the effects of change.

At the present time, ensuring that consistency is maintained among specifications, design, and code is done manually by design reviews and code walkthroughs. Object-oriented programming with inheritance lends itself to a programming style which encourages more reusable code and less code bulk, simplifying code walkthroughs. Additional help is provided by software tools which are available for browsing, organizing, and managing code.

The code generated by the object-oriented approach consists of a number of interacting classes. One of the activities carried out after each prototype build is a careful code review aimed at identifying candidate classes for reuse. Typically, one encounters several similar classes which should be redesigned to have their common functions factored out as a superclass from which they inherit common functionality. Class owners are assigned the responsibility for improving those classes designated for reuse.

The method focuses on target system constraints related to concurrency, which are modeled directly by the actor paradigm as discussed previously. Some preliminary work has been done on the use of event/transaction histories to express timing and spatial constraints, but these notions remain incomplete. Fault tolerance, security, and reliability issues have not been addressed.

The respondents noted that since the semantics for message-passing are the same for actors (which can execute concurrently) and for objects (which cannot), decisions made in the early prototypes with regard to modeling concurrency are easily changed later. Such issues as the number of processors in the target system and the maximum number of tasks per processor are largely ignored until the system is optimized for real-time performance.

The method requires the use of Smalltalk as a PDL and encourages the use of narrative overviews of modules and structured English. It requires the use of message send graphs, inheritance trees, and parts hierarchy diagrams and encourages the use of finite-state, data-flow, entity-relationship, Buhr, and Booch diagrams.

The transitions across phases of the software process are deliberately blurred, to be replaced by a more evolutionary approach which sees the initial simulation incrementally improved through a series of prototypes until it becomes a solution. The respondent cautioned that care be taken to assure that each prototype development is goal-oriented and systematic, and not just the result of thoughtless "hacking". For example, Boehm's spiral model illustrates a systematic approach with the goal of investigating and reducing risk. ABDP does not advocate any particular set of goals, as these are likely to be domain or application specific; it requires only the definition of an objective for each prototype build and criteria for evaluation of the results.

4.1.4 PROJECT CONTROL AND COMMUNICATION

ABDP provides for documentation and quality assurance. It prescribes a fixed format for internal program documentation and tailorable design documentation. It provides a framework for unit/integration testing and requires test planning at one or more precise points in the software process and generation of tests based on system requirements. The respondent reported that organizing the object library to reflect the problem domain facilitated communication and use by applications developers. Since the method provides a working model of the final system early in the development process, the client is able to interact with this model and provide feedback while change is still possible and relatively inexpensive. In effect, the early prototype(s) can be used to validate the client's requirements.

The method assists in the early detection of inconsistencies and/or errors through its evolutionary approach. It focuses on an exploratory programming environment in which requirements tend to be vaguely stated at first, and must be evolved as the project progresses. It begins with a simulation or emulation of the final system, and then, supplying details as required, "grows" it by stages into a solution. Design mistakes are encountered before a large software capital investment has been made. The philosophy is that design mistakes are inevitable when building entirely new systems, and thus it should be relatively inexpensive to recover from mistakes. Optimization is only carried out in the final stages of development when the overall system architecture is stable.

In conjunction with ABDP, a configuration management system called Orwell is being developed to support team programming in large object-oriented applications [Thom88].

4.1.5 EASE OF USE

The ESM testbed has been created and utilized as a project-specific environment. Orwell is being developed to support multiperson Smalltalk programming. The respondent stated that tools are/can be implemented for any personal computer or workstation supporting Smalltalk.

For successful use of the method an experienced developer should have a thorough understanding of object-oriented programming and the basic notions of concurrency, especially synchronization and communication between sequential processes. The concepts of information hiding and abstract data types are essential to the method, while it is compatible with process abstraction, structured programming, inheritance, and module coupling/cohesion. The design approach is derived from Smalltalk, with sophisticated alterations (e.g., delegation replaces inheritance, servers play the role of objects, and workers are identified with methods). The method also incorporates a data analysis system which uses probabilistic reasoning based on Shafer-Dempster belief functions.

The minimum qualifications needed by a development team leader in order to successfully use the method include a bachelor's degree, three to five years of development experience, knowledge of two programming languages, and experience with three to four different software systems.

4.1.6 CONTACT INFORMATION

(Note: The acroynm ABDP is not a standard product name.)

Dr. Brian M. Barry 613-998-2093
Defence Research Establishment Ottawa
3701 Carling Avenue
Ottawa, Canada K1A 0Z4 [Co-developer]

Professor Dave A. Thomas 613-728-1558
Computer Based Information Systems
889 Lady Ellen Place
Ottawa, Canada [Co-developer]

4.1.7 REFERENCES

[Barr87] B. M. Barry, D. A. Thomas, J. R. Altoft, M. Wilson, "Using Objects to Design and Build Radar ESM Systems", Proceedings of ACM OOPSLA '87, Orlando, FL, 1987.

[Thom88] D. A. Thomas and K. Johnson, "Orwell - A Configuration Management System for Team Programming", Proceedings of ACM OOPSLA '88, San Diego, CA, 1988, pp. 135-141.

[CoxB86] B. J. Cox, Object-Oriented Programming: An Evolutionary Approach. Don Mills, Ontario: Addison-Wesley Publishing Co., 1986.

4.2 ADARTS -- Ada based Design Approach for Real Time Systems

4.2.1 BACKGROUND

This method is in beta test; the developer expected it to be released for general use in 1988. ADARTS was developed by Dr. Hassan Gomaa as an extension of his method DARTS, or Design Approach for Real-Time Systems, and is based upon Real Time Structured Analysis and the Naval Reasearch Laboratory Design Method.

4.2.2 DESCRIPTION

ADARTS, Ada based Design Approach for Real Time Systems, is an object-oriented software design method for transforming a specification developed using Real Time Structured Analysis into an Ada based design consisting of tasks and packages.

The key features of ADARTS are its principles for decomposing a real time system into concurrent tasks and packages. Two sets of structuring criteria are used for this purpose. The task structuring criteria, based on the DARTS task structuring criteria, are applied to identify the concurrent tasks in the system. The package structuring criteria, based on the Naval Research Laboratory module structuring criteria supported by Booch's object structuring criteria, are used to identify packages.

4.2.3 TECHNICAL ASPECTS

The developer states that the method is well-suited for applications involving embedded systems or process control, time critical or real-time processing, distributed processing or networks, and image processing or pattern recognition. Specifically, the method is intended to address real-time systems. Data processing or database applications and expert systems or artificial intelligence applications were considered inappropriate for development with the method. Ada is tied to the method as an implementation language; however, ADARTS can be used with other languages as well.

Timing constraints of the target system can be addressed through the use of event sequence diagrams which trace the effect of critical external events through the system. Concurrency issues are handled by means of the method's task structuring criteria.

The method requires a program design language as a textual mode of representation. Required iconographical modes include finite-state diagrams, data-flow diagrams, control-flow diagrams, and Buhr diagrams. The method prescribes mapping rules for translating from data-flow diagrams to task structure diagrams, and from data-flow diagrams to information hiding module structure. The developer states that the method's transfonnation criteria facilitate transformation across phases of the software process.

A number of techniques for clarifying system requirements are encouraged by the method. Data-flow and control-flow analyses are required, as well as design reviews.

By emphasizing information hiding, the method is reported to facilitate incorporation of changes. By requiring design reviews, the method is intended to ensure consistency between specification, design or code. Information hiding modules and modules classified in the module hierarchy are also seen as assisting with identification of potential reusable design components.

4.2.4 PROJECT CONTROL AND COMMUNICATION

ADARTS is compatible with most project management activities; however, it does not address these activities itself.

The client is involved in the development process by means of reviews during specification, and by means of rapid prototyping of the user interface. The majority of documents required to be produced are tailorable in format.

Early detection of inconsistencies and/or errors is assisted by ADARTS by means of design reviews, rapid prototyping, and incremental development.

4.2.5 EASE OF USE

The developer estimated that a development team leader should have as minimum qualifications for successful use of the method a bachelor's degree, three to five years of development experience, working knowledge of three to four programming languages, and experience with three to four different software systems. Major theoretical concepts which should be understood are finite-state machines, concurrent tasks and information hiding. The developer stated that the concepts of information hiding, abstract data-types and structured programming were essential to the method.

4.2.6 CONTACT INFORMATION

Dr. Hassan Gomaa 703-764-6191
George Mason University 703-323-3530
School of Information Technology & Engineering
4400 University Drive
Fairfax, Virginia 22030-4444 [Developer]

4.2.7 REFERENCES

[Goma88a] H. Gomaa, "Extending the DARTS Software Design Method to Distributed Real Time Applications", Proceedings of the 21st Hawaii International Conference on System Sciences, Jan. 1988.

[Goma88b] H. Gomaa, "Adarts - An Ada based Design Approach for Real Time Systems", Software Productivity Consortium Technical Report SPC-TR-88-021, Aug. 1988.

4.3 CAEDE -- Carleton Embedded System Design Environment

4.3.1 BACKGROUND

CAEDE is an experimental design environment for embedded systems, developed by R. J. A. Buhr, G. Karam, and C. M. Woodside at Carleton University. It supports the design method detailed in [Buhr84], an iconic design entry system, a design data base, and design tools. The output of the iconic design process is a design data base of Prolog facts and rules which may be manipulated in various ways by Prolog tools.

4.3.2 DESCRIPTION

The Carleton Embedded System Design Environment, CAEDE, is a design environment built upon integrated Prolog tools for structural, temporal, and performance analysis, as well as partial Ada source code generation.

CAEDE incorporates an iconographic component for design representation. The implemented graphical interface includes primitive icons as well as the capability to represent designer-defined icons. These icons also can be embued with logical properties. The Prolog language is used both to create the design database and as a means for achieving certain goals of the method. These goals include flexibility and extensibility of design as well as providing a way to associate requirements information with design components and their source code translations. According to the developer, Prolog is capable of addressing such goals due to its uniform representation of data and the operations upon data.

Design is considered to have levels and phases. Each level of design may be refined in parallel with other levels of design. At each level, the phases of the design process are:

- Create the structure of the level, to show the partitioning of the design into modules and their interactions.
- Specify the external temporal behavior of each active level, similar, in a way, to specifying the temporal rules for a protocol service specification. Tasks which have standard patterns of behavior are identified by names which imply this standard behavior.
- Specify the internal temporal characteristics required to achieve the temporal interface behavior.
- Provide program strips of code to fill in the gaps left by the skeleton code produced from the previous phases.

4.3.3 TECHNICAL ASPECTS

The developer rated the method well-suited for embedded systems or process control, time-critical or real-time processing, and distributed processing or networks. The method deals with any target configuration and can specifically address concurrency requirements of the target system. The method assumes a class of languages sharing certain features found in Ada. Its principles map directly into Ada constructs.

The method makes use of Buhr diagrams and hierarchical representations in early stages of design. These modes of expression are joined in middle stages by finite-state diagrams, the use of a formal specification language, and a program design language. Use of rapid prototyping is required to clarify the behavior of the system with the software client.

The method uses specification language to generate some of the executable code. Automatic design analysis and execution are intended to bring about early detection of inconsistencies.

The storage of potentially reusable design information in the database was an aspect which the developer judged to contribute towards reusability. The developer stated that the method is not appropriate for use in defining requirements.

4.3.4 PROJECT CONTROL AND COMMUNICATION

CAEDE does not address activities usually associated with project management.

Use of rapid prototyping is required to clarify the behavior of the system with the software client. Documentation associated with the method is provided automatically. The method does not address external documentation standards.

The method provides the following support for verification purposes:

- Prescriptive checking of interfaces;
- Incorporation of an executable model;
- Data- and control-flow analyses of dependencies;
- Use of formal proof techniques;
- Use of a formal specification language;
- Incorporation of tools within itself leading to verification.

4.3.5 EASE OF USE

The developer stated that a project manager could learn the basics of the method in one day. An experienced developer would require five days to learn to use the essentials of the method, with ten days required for a less experienced developer. Expert user level could be attained by an experienced developer in three months.

Overview presentations, classroom tutorials, user manuals, a textbook, periodic technical updates, and on-site consulting are available for training purposes.

The CAEDE research prototype includes tools which provide analysis for finite-state diagrams, Buhr diagrams, and dynamic execution. CAEDE requires a Sun workstation under Berkeley 4.2 Unix. TIMEBENCH is a research testbed based on CAEDE and Machine Charts [Buhr88].

4.3.6 CONTACT INFORMATION

Raymond J. A. Buhr, Professor 613-564-2735
Department of Systems and Computer Engineering
Carleton University
Colonel By Drive
Ottawa, Canada K1S 5B6 [Developer]

4.3.7 REFERENCES

[Buhr84] R. J. A. Buhr, System Design with Ada. Englewood Cliffs, NJ: Prentice Hall, 1984.

[Buhr85a] R. J. A. Buhr, G. M. Karam, and C. M. Woodside, "An Overview and Example of Application of CAEDE: A New, Experimental Design Environment for Ada", presented at International Ada Conference, Paris, France, May 85.

[Buhr85b] R. J. A. Buhr, et al, "Experiments with Prolog Design Descriptions and Tools in CAEDE: an Iconic Design Environment for Multitasking, Embedded Systems", Dept. of Systems and Computer Engineering, Carleton Univ., Ottawa, Canada, Jan. 1985.

4.4 CODE-TOP -- Concurrent System Development Using Transformations of Predicate-transition Nets

4.4.1 BACKGROUND

This method specifically addresses system specification, system design, system implementation and software quality assurance. It is founded upon an object-oriented approach, formal specification of concurrency by predicate-transition nets, and abstract machine hierarchy. The developer reported the method well-suited to the operational model, transformational model, and verification by proofs paradigm.

The principal architects of the method are Patrick de Bondeli and two companies: Aerospatiale (Space Division) and CR2A. The method is in development and is targeted for release for general use without tools in 1989; with tools, in 1995.

4.4.2 DESCRIPTION

CODE-TOP, or COncurrent system DEvelopment using Transformations Of Predicate-transition nets, is principally based on the use of Pr-T nets, a high-level class of petri nets, and covers the complete development cycle for the software of a concurrent system. The first step is the development of a formal specification from the informal requirements provided by the client and the last step is the production of executable code (Ada presently) implementing this specification.

The system specification, resulting from the first step, is composed of Pr-T nets giving the overall structure and concurrency properties, and of pre/post-condition systems specifying the sequential algorithms. (On the Pr-T net, the execution of a sequential algorithm only appears as a function or procedure call in the conditional or operative part of a transition, and this function or procedure is separately specified by a pre/post-condition system). The data domains used on the Pr-T nets and in the sequential function or procedure specifications are described, as far as possible, in terms of the implementation language (in practice presently, Ada is used with extensions to make the semantic domains more precise). If the system is complex enough, the specification will be split into different views or subsystems. This partition of the specification may require the use of a requirements analysis technique which is not an integral part of the method.

After the system specification step comes the high-level design step which consists of partitioning the system (or each subsystem) into smaller units. An object-oriented approach is used here and the units are structured as abstract data types and objects. An object is a logically coherent data structure which is specified by the resources and operations which it exports to its users. The object presents to its users an abstract specificaiton of these exported resources and operations and hides the implementation details from them. An abstract data type is a class of objects factoring the common properties of the objects in the class.

The specification of each object or abstract data types will include:

- an Ada package (or generic package) specification giving the syntax and static semantics of the object or abstract data type;
- a dynamic semantics specification including 1) Pr-T nets composed from state-machines representing the operations, the concurrently usable resources and their states, and the initialization sequence (when relevant), and 2) pre/post condition systems describing the sequential, algorithmic part of the operations. These condition systems are included as comments in the Ada package specification.

At this step the developer is strongly encouraged to verify that the set of specifications of abstract data types and objects produced thus far makes a correct partition of the system or subsystem, preserving its desired properties.

The detailed design step consists of implementing each object or abstract data type in terms of structures of the implementation language and use of lower-level objects and abstract data types which will have to be specified using the same techniques as for higher-level objects and abstract data types. During this step, Pr-T net transformations are used to transform the abstract Pr-T nets which are components of the specifications into more concrete Pr-T nets which are made from elementary net structures modelling the structures of the implementation language (Ada). Each implementation of an object or abstract data type is proven correct, either de facto because only predefined correct transformation rules were used to derive this implementation, or explicitly if the developer had to incorporate specific transformations.

Several iterations of the detailed design step may be necessary; each iteration then consists of implementing an object or abstract data type while defining or completing the specification of the lower-level objects or abstract data types which are used in this implementation. During this process objects and abstract data types are grouped into "abstract machines" which are built in such a way as to have the use relations between machines form strict tree structures (there can be several hierarchies, each one forming a tree structure, sharing common nodes). The developer states that this hierarchic structure makes the system easier to comprehend than a "flat" object structure.

4.4.3 TECHNICAL ASPECTS

CODE-TOP is intended to address all concurrent systems that can be implemented in a procedural way. The developer rated it well-suited for embedded systems or process control, time-critical or real-time applications, systems programming, distributed processing or networks, and large-scale simulation or modeling. It was considered inappropriate for developing applications for expert systems or artificial intelligence and image processing or pattern recognition. Only Ada has been seriously worked out as an implementation language, but the developer included LTR3, MODULA-2, and PORTAL as other convenient implementation languages existing today which support data abstraction, procedural development, and concurrent structures.

The method requires narrative overviews of modules, a formal specification language, and mathematical notation as textual modes of representation. Iconographical modes required are petri nets and abstract machine diagrams. The form of petri nets used are called Pr-T (predicate-transition nets), which support the specification of concurrent properties of the system.

The method prescribes mapping rules for translating from one mode of expression to another. Diagrams representing a hierarchy of abstract machines composed of abstract data types and objects are translated into an Ada package (or LTR-3 or MODULA-2 modules structures). Detailed Pr-T nets describing the structure of the implementation of abstract data types and objects are translated into Ada concurrent control structures.
Other mapping rules are prescribed at different steps of the method, but they are of a different nature in that they stay within the same mode of expression and are semantic transformations within that mode.

In facilitating the transformation across phases of the software process, the method can be seen as an extension of the "transformational approach" from sequential software development to the domain of concurrent systems. To go from specification to design, the developer is encouraged to use correct transformation rules, provided within the framework of the method, whenever they are applicable and to provide proof schemes for his own specific transformations when no pre-existing transformation rule seems applicable. The passage from detailed design to an Ada code frame is a pure translation process (no semantics are added) using rules provided within the method. The implementation of sequential algorithms is not addressed by the method, in order to leave the developer free to choose any of the already existing methods for such.

Analysis and review techniques essential to the method are data-flow analysis, control-flow analysis, formal proof techniques, and design reviews. The method aims to incorporate changes in the requirements in an orderly and secure way, by requesting that the different steps be processed again using the systematic rules such steps incorporate. The method strictly requires that a change be made first at the highest applicable level and that the development cycle be iterated again from this level using the rules provided by the method, thus assisting in ensuring consistency within the software product.

The use of pre-existing correct transformation rules can be seen as a form of reuse (of their correctness proofs and target schemes).

4.4.4 PROJECT CONTROL AND COMMUNICATION

Project management activities are not addressed by the method.

When the specifications are developed from the requirements, the requirements generally reveal inconsistencies and incompleteness which should be corrected in cooperation with the client. This development step should normally end by having the client approve the specificaitons as conforming to the requirements. However, if this agreement is too difficult to reach because the system is complex or the requirements are imprecise, protyping demonstrations can help the client to make his needs more precise. CODE-TOP does not provide any specific help for prototyping but recognizes that such a step may be useful for these cases.

Regarding documentation required by the method, the functional specification, behavioral specification, and interface specification are merged and reported as a functional specification. This specification uses formal notations; however, the format is tailorable within the method since the way the component specifications are organized depends upon the nature and complexity of the system. The method has a fixed format for system structure chart and design document.

The method assists in the early detection of inconsistencies by strongly encouraging correctness proofs to be processed or correct transformations to be used at each development step beyond the requirements/specification step. Test planning, generation of tests, unit/integration testing, and field or acceptance testing are required but directions for accomplishing such are not provided by the method.

4.4.5 EASE OF USE

The developer felt that an experienced developer should understand the following theoretical constructs for successful use of the method: formal specification techniques, concurrent systems, petri nets including high-level nets (Pr-T nets), object oriented development, and the transformational approach to software development.

Minimum qualifications for a development team leader for using the method successfully were given as an advanced degree, three to five years of development experience, knowledge of the target language (Ada presently), and experience with two different software systems (only the software development environment system and the target machine system are strictly necessary).

The method is not yet supported by any tools; however, the developer expects that professional workstations (such as the SUN-3 or SUN-4) with a graphic processor and state-of-the-art object management system (such as PCTE) can provide a good host environment for future tools.

4.4.6 CONTACT INFORMATION

Patrick de Bondeli 33 1 47689797, ext. 4482
CR2A (Conception et Realisation d'Applications Automatisees)
19 avenne DuBonnet
92411 Courbevoie
Cedex FRANCE [Developer]

4.4.7 REFERENCES

[deBo83] P. de Bondeli, "Models for the Control of Concurrency in Ada Based on Pr-T Nets", Proceedings of the Adatec-Ada Europe Joint Conference on Ada, edited by the Commission of the European Communities, Brussels, March 1983.

[deBo86] P. de Bondeli, "On the Use of Semantic Specification for the Verification and Validation of Real Time Software", Proceedings of the 3rd IDA Workship on Ada Verification, Research Triangle Institute, Research Triangle Park, NC, May 1986.

In addition, the developer has several other papers both presented at other conferences as well as awaiting publication. A manuscript of a paper to be published in a book on specification techniques for concurrent systems (to be published by Ada-Europe in the CUP Ada Companion Series) is available on request from the developer.

4.5 COMIC -- Conceptual Modelling and Information Construction

4.5.1 BACKGROUND

The principal architect of this method is Hannu Kangassalo. Presently in development, the method is anticipated for release for general use in late 1989.

Founded on object-oriented and knowledge engineering approaches, this method prescribes specific procedures for system design, system implementation, and data base use. The developer considers the method well-suited for use with the incremental and 4GL software process paradigms.

4.5.2 DESCRIPTION

COMIC, Conceptual Modelling and Information Construction, begins with the collection of concept definitions from the application domain in the form of graphical concept structures. This step is expected to be accomplished by the clients, but persons familiar with the method may provide support.

These concept structures are checked and transformed into elementary conceptual schemas (ECS); then, the ECS's are integrated into a common conceptual schema.

From the final conceptual schema the relational data base schema is derived for the application data base. The computer-supported translator makes the mapping between two schema levels. The relational data base schema is inspected and data is fed into the data base.

After the above steps, the development support system is ready for use. A user can make queries from the data base from the level of the conceptual schema. This is accomplished by pointing at those concepts on the conceptual schema from which data is desired and by giving a process command. Updates can also be made from this conceptual schema level.

4.5.3 TECHNICAL ASPECTS

COMIC is intended to address office automation and data base processing applications. The developer rated the method inappropriate for use with embedded systems or process control, scientific or engineering applications, systems programming, image processing or pattern recognition, and large scale simulation or modeling. SQL is the implementation language which is tied to the method.

The method requires the use of a formal specification language, Warnier/Orr diagrams, and conceptual schema for textual modes of representation. Entity-relationship diagrams, concept structures, conceptual schema, and relational schema are required for iconographical representation. The developer indicated that other modes of expression may be useful in some cases as well. The method prescribes mapping rules for translating from one mode of expression to another. The concept structure diagram is mapped to the conceptual schema, the conceptual schema is mapped to the relation schema, and a graphical query is mapped to SQL. The method is seen to facilitate the transformation across phases of the software process by automatic, interactive translation from specification to design (data base schema) and by automatic translation of data base queries and updates to SQL, which are given to the data base management system.

Incremental or evolutionary development and knowledge engineering are required to clarify system requirements or behavior. The method requires concept analysis and in some cases data-structure analysis.

New requirements are defined as extensions to the conceptual schema, from which new data base schema will be generated, if necessary. Data base translation must be done in some cases. Consistency is maintained in the developing system by allowing changes to be made only in the specification, from which design and code are generated automatically.

The method addresses reuse by incorporating a concept definition library and conceptual schema library.

4.5.4 PROJECT CONTROL AND COMMUNICATION

The software client is involved in the software development process by being expected to make concept definitions (concept structures) and to check the resulting conceptual schema.

The method requires that a number of documents be produced; format prescription and degree of automated support vary. The system structure chart and conceptual schema have fixed formats and are automatically generated based on data produced from other steps in the method. The data base schema and data dictionary are likewise automatically generated.

The developer states that the method assists in the early detection of inconsistencies. This is the result of the graphical concept definitions improving the understanding of the concepts of the problem domain, and the graphical conceptual schema helping one to visualize the definition of the application field. The method also prescribes specific procedures and provides automated support for the prescriptive checking of interfaces.

4.5.5 EASE OF USE

Essential concepts of COMIC are stepwise refinement, abstract data-types, and inheritance. The concept of module coupling/cohesion was considered inconsistent with the method.

In the developer's opinion, a development team leader would need, for successful use of the method, a minimum of a bachelor's degree, one to two years of development experience, knowledge of one programming language, and experience with one software system.

Appropriate hardware/software configurations for hosting the tools of the method include Apollo DN 3000 + GMR + Pascal, Oracle DBMS, and M-Prolog.

4.5.6 CONTACT INFORMATION

Hannu Kangassalo
University of Tampere
P.O. Box 60SF-33101 Tampere
FINLAND [Developer]

4.5.7 REFERENCES

[Kang83] H. Kangassalo, "CONCEPT D - A Graphical Formalism for Representing Concept Structures", in H. Kangassalo, ed., Second Scandinavian Research Seminar on Information Modelling and Data Base Management. Acta Universitatis Tamperensis, Ser. B, Vol. 19, Univ. of Tampere, Finland, 1983.

[Kang84] H. Kangassalo and P. Aalto, "Experiences on User Participation in the Development of a Conceptual Schema by Using a Concept Structure Interface", in B. Shackel, ed., INTERACT '84. Proceedings of the first IFIP Conference on Human Computer Interaction. North-Holland, Amsterdam, 1984.

[Kang88] H. Kangassalo, "CONCEPT D - A Graphical Language for Conceptual Modelling and Data Base Use", Proceedings of the 1988 IEEE Workshop on Visual Languages, University of Pittsburgh, Oct. 10-12, 1988.

4.6 ERAE -- Entity Relationship Attribute Event

4.6.1 BACKGROUND

An approach to requirements engineering based on an expressive and formal language, ERAE, together with its accompanying methodological guidance, has been developed under the Commission of the European Communities, Project 432 (Meteor) of the ESPRIT program. The principal architects are E. Dubois, J. Hagelstein, and A. Rifaut. This method, in development and targeted for release in 1991, is an extension of entity-relationship modeling. Specific procedures are provided for requirements definition or clarification and system specification.

4.6.2 DESCRIPTION

ERAE, Entity Relationship Attribute Event, is a method for requirements engineering, i.e., for the identification of those aspects of a computer system which are relevant to the user. The method is based on the following main assumptions:

- requirements should not first focus on the computer system, but rather on a more general system formed by the computer and its environment;
- requirements should be unambiguous;
- the validation of requirements may benefit from powerful analysis techniques, such as the identification of consequences of a set of requriements.

The ERAE language supports these assumptions by providing concepts suited for modelling actual situations, rather than just computer system behaviors. The developers call this the "closed system specification" paradigm, and state that the most effective use of the method is dependent upon this paradigm. The ERAE language is formally-defined and supports rigorous deduction rules.

There are two parts to the ERAE language. The graphical part describes the vocabulary of the application by identifying various categories of concepts: entities and events, organized in classes or considered individually, relations, attributes, and so on. A second part of the language states the known or desired properties of these concepts. It consists of first order logic statements, with the previously-defined vocabulary occurring as functions, predicates and sorts. This logic is actually a temporal variant of first order logic, allowing one to express invariant properties (possible states) as well as dynamic properties (possible evolutions), in either an operational or declarative style.

At the strategic level, the ERAE method recommends that one first identify objectives, and then to define an environment and a computer system whose cooperation meets the objectives. At the tactical level, the method prescribes a repeated cycle of interaction with clients and analysis of the evolving requirements. The latter heavily relies on the inferencing capabilities of the language. Examples of analysis techniques are consistency checking, generation of variant forms of statements, and generation of consequences. Heuristic rules are provided to help select the proper analysis technique at any step. Interaction with non-analysts involves the conversion of fragments of the specification into more appropriate forms, e.g., diagrams, tables, natural language.

4.6.3 TECHNICAL ASPECTS

The method is not tied to a particular implementation language. Application areas considered well-suited for using the method are embedded systems or process control and time-critical or real-time systems. The method is intended to address real-time embedded systems in particular.

ERAE can express several requirements of the target system. The use of real-time temporal logic allows one to convey real-time and performance constraints. Requirements on the environment and security requirements can be expressed in the language, including forbidden situations or behaviors. "Events" are used to describe the interface requirements, and simultaneous events address concurrency.

Textual representations used by the method include a formal specification language and mathematical notation. Iconographical modes include entity-relationship diagrams. The method provides mapping rules for translating from the graphical part of the language to the textual part. Facilitation of transformation across the phases of the software process is not particularly addressed, since the method is limited to requirements definition and produces a formal specification to be used as a starting point for design. However, the developer states that the conversion from the ERAE specification to a design language can be automated if the latter is formal and expressive enough. This has been done for the design language COLD, used at Philips Research Laboratory Brussels.

A primary analysis technique utilized by the method is consequence analysis, in which rules are given to deduce consequences from the known facts. Formal proof techniques and data-structure analysis are also used. The method strongly encourages incremental or evolutionary development in order to clarify system requirements or behavior.

The method assists in reducing efforts needed to incorporate changes in the requirements by the fact that the ERAE language is declarative, which means that facts are stated independently of each other. A change only affects a limited number of statements. In addition, the method provides deduction rules for analyzing consequences. This allows for the investigation of the consequences of changes.

4.6.4 PROJECT CONTROL AND COMMUNICATION

Activities associated with project management are not addressed.

The software client is involved in the software process during requirements definition, in the form of meetings where different interviewing techniques are used. The technique which is most specific to the method is the validation of earlier statements through the consequences of those statements.

The documents required to be produced consist of requirements definition, functional specification, behavioral specification, and interface specification. Fixed formats are prescribed by the method for all these documents.

Assistance in early detection of inconsistencies or errors is given by the method in that the method stresses the detection of mistakes during requirements specificaiton. Deductions (in a formal logic) are used to check consistency and to generate consequences of the requirements gathered so far. The client is asked to validate these consequences.

Verification is based on proof techniques; testing activities are not addressed by the method.

4.6.5 EASE OF USE

Concepts essential to the method are stepwise refinement, information hiding, and inheritance. First order logic should be understood by an experienced developer in order to successfully use the method. In the opinion of the developer, minimum qualifications of a development team leader would include an advanced degree and one to two years of development experience on one software system for successful use.

Sun workstations are appropriate for supporting graphical and textual editors and syntax checkers for use with the method.

4.6.6 CONTACT INFORMATION

J. Hagelstein +32-2-674 22 42
Philips Research Laboratory
2, av. Van Becelaere
B-1170 Brussels
Belgium [Developer]

4.6.7 REFERENCES

[Dubo88] E. Dubois, J. Hagelstein, and A. Rifaut, "Formal Requirements with ERAE", Philips Journal of Research, V. 43, 3/4, 1988, pp. 393-414.

[Hagl88] J. Hagelstein, "Declarative Approach to Information Systems Requirements", Knowledge Based Systems, V. 1,4, 1988, pp. 211-220.

GANDALF

4.7 GANDALF -- Gandalf System for Structure Oriented Environment Generation

4.7.1 BACKGROUND

Named after the benevolent wizard in J.R.R.Tolkien's fantasy series "The Lord of the Rings", GANDALF has been a continuing research project into the generation of software development environments since 1976. N. Haberman started the initial project management work with design, development and maintenance of the System (or Software) Development Control system (SDC). GANDALF's research in system version control environments started concurrently with SDC, followed by P. Feiler's development of the Language-Oriented Incremental Programming Environment (LOIPE).

[Habe86] describes related work (in this project and others) on software development environments and their components, including Medina-Mora's A Language Oriented Editor (ALOE), the Cornell Program Synthesizer, Interlisp, Smalltalk, Mesa/XDE, Cedar, the Synthesizer Generator, Mentor, PDE, Syned, Pecan, Toolpack, Magpie, Cope, Poe, R^{n}, DICE, DOSE, UNIX, Cades, the Stoneman requirements for an Ada programming environment, Arcturus, SAGA, and IPSEN.

Although Gandalf is not commercially marketed, it has been in use for over ten years. The current version is a functioning instantiation of the research ideas of the Gandalf project. Its developers have made it available upon request to industrial and academic organizations. This research has been supported in part by the Software Engineering Division of CENTACS/CORADCOM (now called CECOM), Fort Monmouth, NJ, by ZTI-SOF of Siemens Corporation, Munich, Germany, by the Defense Advanced Research Projects Agency (DARPA), and other organizations. For a more detailed discussion of the history, see [Notk85].

4.7.2 DESCRIPTION

The Gandalf System for Structure Oriented Environment Generation provides for the semi-automatic creation of families of project-oriented software development environments that incorporate both programming and system development environments. As described in [Habe86], "Gandalf programming environments are language-oriented editing systems that support source-level debugging and either interpretation or incremental compilation (with associated incremental linking and loading)." Gandalf system development environments support version control (related to the notion of programming-in-the-large [DeRe76]) and project management (related to "programming-in-the-many"). Gandalf software development environments are intended to be characterized by integration, uniformity, interactive user interfaces, and a well-defined system state.

A Gandalf software development environment is integrated through the use of a common development database. The shared information, coupled with the ability of the language-oriented editors to perform both syntactic and semantic analysis, can avoid unnecessary recompilation. Action routines implement the specific run-time behavior that the designer might want to realize in the environment being created. Action routines can be used for various purposes, including checking of semantics, window and memory management, and keeping the system being developed in a well-defined state. Each entry in the database has a "daemon" associated with it according to the datatype of that entry. These daemons monitor all operations on the database entries. When a daemon observes a relevant database operation, it can become active to perform semantic checks, automated operations, function as an interpreter for dynamic semantics, etc.

Gandalf utilizes 4GL techniques. The respondent stated that it is also well-suited for use in conjunction with rapid prototyping and evolutionary development. It prescribes specific directions and procedures for performing the activities of system specification, system implementation, and version and access control. Founded upon object-oriented and event-oriented approaches, it is also compatible with a data structure-oriented approach, entity-relationship modeling, and functional decomposition.

The most recent release, in March, 1989, incorporated TransformGen [Garl88], a component which generates automatic converters for structure oriented environments when the environment itself undergoes a new release. Continuing research has been into instantiation of a configuration management aspect to create multi-user programming environments.

4.7.3 TECHNICAL ASPECTS

Several Gandalf environments have been created and used. These include: the Gandalf Prototype, a complete Gandalf software environment; GNOME, a set of programming environments used by over 700 students per semester at Carnegie-Mellon University; and SMILE, a production-quality internal development tool of the Gandalf project. The respondent stated that the technique is intended to address the area of interactive environments and well-suited for use in data processing/database applications, and expert systems/artificial intelligence. It is also compatible for use in scientific programming, systems programming, and large scale simulation.

Although it is not tied to a required implementation language, it has been used in conjunction with Pascal, Ada, and C. It requires the use of a formal specification language. Gandalf produces language-oriented editors that can prompt the user with standard language constructs, and detect syntactic and semantic errors in source code.

Gandalf facilitates the software development process by generating integrated environments that support teams of programmers working on large projects. Programming environments provide assistance at the programming level through such utilities as language-oriented editors, source-level debuggers, and incremental compilers or interpreters, linkers, and loaders. System development environments provide system version control to track dependencies of the components of a system and support project management objectives, such as avoiding updates to previous versions.

Although there is work in progress on generating multi-user environments, the current system provides single user environments and does not protect against multiple concurrent update activity. Generation of the environments themselves can be done on a multi-user system, utilizing the protection of the host operating system.

Gandalf encourages the use of rapid prototyping, evolutionary development, and executable specifications. It does not address reusability of the applications software, but rather reusability of the components of the software development environment.

Since the system generates the code based upon specifications, maintaining the consistency of specifications, design, and code of evolving applications software is automatic. However, a problem for programming environments has been that information maintained by an environment becomes invalid when the environment is replaced by a new release. TransformGen monitors changes made to the structure definition of a database and automatically generates transformers that will convert existing databases to the new description. The process is completely transparent to the target user.

4.7.4 PROJECT CONTROL AND COMMUNICATION

The environments produced by Gandalf provide project management support and system version control support. Project management support includes reserve, deposit, and creation of history logs. System version control support includes parallel and successive versions and automatic system generation.

The tools being created are generated from high-level specifications. This facilitates prototyping of environments for review by the end-users of the Gandalf environments, i.e., the applications developers.

4.7.5 EASE OF USE

For successful use of the method, a development team leader would need two to three years of college-level technical education, one to two years of development experience, a working knowledge of one programming language, and development experience with one other system. An experienced developer should have an understanding of BNF grammars, hierarchical databases, and active databases. The concept of genericity is essential, while the concepts of stepwise refinement, abstract data-types, module coupling and cohesion, and use of assertions are compatible.

The main hardware/software requirement is that the machine run BSD UNIX or a compatible such as ULTRIX, MACH, or the Sun OS. Limited support is provided for windowing systems such as SUNTOOLS or X Windows. This support is being enhanced to better utilize the workstation environment. The system is currently being ported to System/V.

4.7.6 CONTACT INFORMATION

David B. Miller (412) 268-2857
Carnegie Mellon University
Schenley Park
Pittsburgh, PA 15213 [project manager]

A. Nico Habermann
Dean, Department of Computer Science
Carnegie Mellon University [principal investigator]

4.7.7 REFERENCES

[DeRe76] R. DeRemer and H. Kron, "Programming-in-the-large versus programming-in-the-small", IEEE Transactions on Software Engineering, Vo. SE-2, June 1976, pp. 80-86.

[Garl88] D. Garlan, C. Krueger, and B. Staudt, "TransformGen: Automating the Maintenance of Structure-Oriented Environments", Carnegie Mellon Technical Report No. CMU-CS-88-186, 1988.

[Habe86] A.N. Habermann and D. Notkin, "Gandalf: Software Development Environments", IEEE Transactions on Software Engineering, Vol. SE-12, No. 12, Dec. 1986, pp. 1117-1127.

[Krue88] E.W. Krueger and A. Bogliolo, "Scaling Up: Segmentation and Concurrency in Large Software Databases", Carnegie Mellon Technical Report No. CMU-Cs-87-178, Feb. 1988.

[Notk85] D. Notkin, "The GANDALF Project", The Journal of Systems and Software, Vol. 5, No. 2, May 1985, pp. 91-105.

4.8 GOOD -- General Object-Oriented Software Development

4.8.1 BACKGROUND

This method is in the process of development by the Software Engineering Laboratory at NASA's Goddard Space Flight Center. Its principal developers are Ed Seidewitz and Mike Stark. GOOD has been influenced by Grady Booch's Object-Oriented Design, George Cherry's PAMELA, and Structured Analysis/Structured Design.

4.8.2 DESCRIPTION

GOOD, General Object-Oriented Software Development, prescribes specific procedures for conducting requirements definition, system specification, system design, and system implementation or installation. It is founded on an object-oriented approach and entity-relationship modeling. Although not dependent on any particular software process paradigm, the developer rated the method well-suited within the context of Boehm's spiral model and the incremental model. The method is in development.

GOOD begins system specification by identifying the entities in a problem domain and their interrelationships. This follows from the notion that the modules of an object-oriented design primarily represent domain entities, not just functions. Entity-relationship and data flow techniques can then complement each other, the former delineating the static structure problem domain and the latter defining the dynamic function of a system.

In object-oriented design, the unit of modularity is the object, which is a package of data and operations on that data. Graphical object diagrams represent the design of a system of interconnected objects. The developer states that the system design process is guided by principles of abstraction, information hiding, composition, and seniority.

Abstraction analysis is the process of making a transition from the structured specification to an object-oriented design. Abstraction analysis involves finding the central entity, that is, the set of processes and data stores which are the most abstract. After the central entity is identified, processes and data stores which directly support it are identified. These steps are repeated until all processes and data stores have been associated with each new entity. The result of abstraction analysis is an entity graph which displays the highest abstraction possible and all interconnections between entities. This entity graph is the starting point for object identification. At this stage the designer's judgement is required to tradeoff between design complexity, object abstraction and control hierarchy. Once objects have been identified and object diagrams drawn, the next step is to identify the operations provided and used by each object to produce object descriptions.

4.8.3 TECHNICAL ASPECTS

GOOD was rated well-suited to scientific or engineering applications, systems programming, and large scale simulation or modeling. The method is not intended to address a specific type of applicaiton area, however. Although the method was developed for use with Ada, it has general applicability to other languages, especially to languages with modular package, object-oriented and/or abstract data type constructs.

Required textual modes of representation are narrative overviews of modules, program design language, and object descriptions. Iconographical modes required are finite-state diagrams, data-flow diagrams, control-flow diagrams, entity-relationship diagrams, and object diagrams. Mapping rules are prescribed for translating from entity-relationship/object-oriented data flow diagrams to object diagrams, and from object diagrams to compilable design. For transitioning between phases, the developer states that the life cycle, object-oriented philosophy of the method allows the statement of clear rules and heuristics.

Data-flow and control-flow analyses are required techniques, and data-structure analysis, design reviews, and code walk-throughs are strongly encouraged. Incremental or evolutionary development is also strongly encouraged as a means of clarifying system requirements or behavior.

The method's assistance in producing a consistent object-oriented framework provides a baseis for full traceability from specifications to design to code. The guidelines provided for reducing module coupling by promoting a strict dependency hierarchy are seen to assist in the identification of possible reusable components. Also provided are heuristics and techniques for creating and using generic, reusable modules.

4.8.4 PROJECT CONTROL AND COMMUNICATION

GOOD prescribes specific procedures and directions for test planning at one or more precise points in the software process, unit/integration testing, and field or acceptance testing. It provides a framework for conducting other testing activities. Involvement of the client is mainly during system specification and at various reviews.

The method does not address project management activities involving risk or cost assessment; further information on other management activities was not provided.

Required documents of the method are requirements definition, functional specification, behavioral specification, data dictionary, design document, and user manual. The formats for the above documents are not prescribed.

4.8.5 EASE OF USE

Concepts regarded by the developer as essential to the method are information hiding, abstract datatypes, genericity, inheritance, and module coupling/cohesion. Successful use of the method by an experienced developer would also include an understanding of objects and classes, state machines (not necessarily finite state machines), and entities and relationships. In the opinion of the developer, a development team leader would need as a minimum a bachelor's degree, three to five years of development experience, working knowledge of two programming languages, and experience on three to four different software systems in order to use the method successfully.

4.8.6 CONTACT INFORMATION

Ed Seidewitz — 301-286-7631
Code 554.1
Goddard Space Flight Center
Greenbelt, MD 20771 — [Developer]

GOOD

4.8.7 REFERENCES

[Seid86] E. Seidewitz and M. Stark, "General Object-Oriented Software Development", Goddard Space Flight Center Software Engineering Lab Document SEL-86-002, Greenbelt, MD, Aug. 1986.

[Seid87a] E. Seidewitz and M. Stark, "Towards a General Object-Oriented Ada Life Cycle", Proceedings of the Joint Ada Technology Conference/Washington Ada Symposium, March 1987.

[Seid87b] E. Seidewitz and M. Stark, "Towards a General Object-Oriented Software Development Methodology", SIGAda Ada Letters, Vol. 7, No. 4, July-Aug. 1987, pp. 54-67.

[Seid88] E. Seidewitz, "General Object-Oriented Software Development; Background and Experience," Proceedings of the Hawaii International Conference on Systems Science, Jan. 1988.

4.9 HCDM -- Hierarchical CHILL Design Method

4.9.1 BACKGROUND

HCDM is a graphical software design method for distributed, messsage-based real-time systems. Developed by M. Fastenbauer, T. A. Wolf, H. Saria, and G. Hoczer, it adapts well-known paradigms from SADT and SDL, other design methods, and the design habits and experiences of Alcatal N.V. to provide a consistent framework for design and implementation of telecommunication and process control software.

GSDS provides the computer support for HCDM, consisting of a central design database and tools. Because documentation and code are generated out of the same source there is always a threefold consistency between design, code, and documentation.

The method is in development and anticipated to be released for general use in 1989.

4.9.2 DESCRIPTION

Hierarchical CHILL Design Method, HCDM, covers three aspects: design, documentation, and code. The design with HCDM spans the whole range from top level design down to detailed design with visual programming. Prescribing specific directions and procedures for conducting system design and implementation, it requires more effort in the design phase, but permits omission of the process of coding and producing documentation (to some extent). It incorporates sufficient detail in the design to allow the automatic generation of both CHILL code and corresponding documentation. In the developers' opinion, the total effort measured over the whole development cycle will be approximately the same. but the quality of the results will be considerably higher.

Using an approach founded upon CCITT's Specification and Design Language (SDL) and functional decomposition, HCDM also incorporates data-flow, control-flow, and event-oriented approaches. Falling within the context of an operational model of the software process, it is well-suited for use with the Waterfall model and may be used in conjunction with rapid prototyping and 4GL database systems.

In HCDM systems are described by four models:

- the Static Model,
- the Dynamic Model,
- the Data Model, and
- the Physical Model.

In the Static Model the system is divided into functional units, the so-called function blocks. These function blocks can be connected by directed channels that indicate the communication abilities of the function blocks. Only connected function blocks may exchange messages in the direction given by the channel. Each channel has an associated set of messages that can be sent from the source block to the destination block. Using functional decomposition, each function block at one level can be refined on the next level. Channels and the messages that reside on these channels will also be decomposed. Thus, each level of a HCDM design represents a complete desciiption of a system at a specific level of abstraction.

The function blocks of the deepest decomposition level represent processes. Their behavior is described by the Dynamic Model. This model represents the dynamic behavior by means of states and state transitions triggered by messages. During state transitions, messages can be sent and some actions can be performed. This is described in two parts: by one Finite Message Machine (FMM) Overview Diagram and several Transition

Diagrams (one for each FMM state). As opposed to the SDL process diagrams, the Transition Diagrams of HCDM have the CCITT High Level Language (CHILL) semantics, so the developer states that the transformation of a HCDM design to CHILL code is almost a straightforward task.

The Data Model's intent is to give a description of all relevant data while the Physical Model is expected to contain information on hardware partitioning and configuration parameters. The current Data Model implementation is only a partial one. Only global data such as Message Lists and the Database Charts are handled within GSDS; variables, types, etc. have to be written in CHILL syntax using separate files. Physical aspects are handled only implicitly and are not clearly distinguished in the current version of GSDS.

The computer support, GSDS, includes a central design database, graphics and textual editors, a consistency checker, documentation generator, and program generator. It provides for command procedures, so that code and documentation can be generated by applying a single command. To make it possible for the program generator to create complete (compilable, executable) and efficient code, detailed information has to be specified within GSDS (e.g., message priorities, message data, access rights to the database, etc.). Much of the input is made graphically. Textual input is only necessary for information where diagrams do not increase the clarity (e.g., Message Lists). All information is stored in the design database and all tools get their information from it.

4.9.3 TECHNICAL ASPECTS

Developed to address telecommunication, switching, and railway control applications, HCDM is oriented towards parallel, distributed systems in general and CHILL systems in particular. It is also well-suited for use in embedded systems, process control, and real-time systems. HCDM allows the modeling of message communication, parallel processes, time out control, and relational real-time database.

HCDM requires the use of PDL, FMM Overview Diagrams, State Transition Diagrams, Subtransition Diagrams, Static Diagrams, SDL diagrams, Message Lists and Database Charts. FMM Overview Diagrams are a type of finite- state diagram. Within the dynamic model, the "finite message machine" refers to finite automata with asynchronous message communication and actions performed during state transitions. Static Diagrams are similar to data-flow diagrams. These structure diagrams show function blocks, the communication channels between these function blocks, and their hierarchical decomposition. A Message List is a collection of all the message definitions at a design level. A Database Chart consists of definitions for the database. Subtransition Diagrams are extracts of Transition Diagrams with one entry and zero or one exit.

The method provides mapping rules for translating from:

- Static Diagrams to a block hierarchy,
- Static diagrams to Finite Message Machines,
- Finite Message Machines to Transition Diagrams.

It facilitates transformations across phases, by taking the designer from top level design to such a level of detail that both code as well as documentation can be generated automatically by GSDS. This is achieved by taking as much information as possible from each model into the other ones.

The techniques of dynamic animation, simulation, and executable specifications are strongly encouraged. Such facilities have not been developed for HCDM, but the developer states that it would be possible to do so. The method also encourages data-flow and control-flow analysis.

With HCDM, all changes are applied to the design of the system. There is no need to change documentation or code independently, since it is always possible to derive them automatically. Thus, the design, documentation, and code remain consistent.

There is some implicit support of identification of reusable elements in the form of clarity of design. Component parts of a system can be easily identified together with their interfaces due to the qualities of the several models.

4.9.4 PROJECT CONTROL AND COMMUNICATION

HCDM expects there to be a pre-existing requirement or analysis model of what the client wants. There is no automated or formally defined connection to such a model. However, functional decomposition and stepwise refinement coupled with graphical representations of both structure and behavior allow that the system design can be presented at different levels of detail in comprehensive form to the customer.

The method requires project management and provides a mechanism for access rights for designers.

Documentation is tailorable and automatically generated based on data produced from other steps in the method. This includes behavioral design, system structure, finite message machines, transition diagrams, and cross references.

The method assists in the detection of inconsistencies or errors through application of rules which are checked by a special utility, the consistency checker. The checker also takes care of simple rules of completeness and exclusion. The method requires unit/integration testing.

4.9.5 EASE OF USE

For successful use of the method, an experienced developer should have an understanding of finite state machines and communication between asynchronous, parallel processes. The method utilizes the techniques of stepwise refinement and process abstraction; it is inconsistent with the techniques of information hiding, abstract data-types, structured programming, use of assertions, and module coupling/cohesion.

The minimum qualifications needed by a development team leader are a bachelor's degree, one to two years of development experience, knowledge of one programming language, and experience on one other software system.

The support environment for GSDS consists of a VAX/VMS host, a Tektronix 4207 terminal, the Tektronix Graphic package PLOT 10 IGL, and the DEC DBMS or Oracle. During the migration period from the DEC DBMS over to relational Oracle, a SQL-based DBMS, both are required.

4.9.6 CONTACT INFORMATION

Thomas A. Wolf — 011-43-1/39 16 21
Alcatel Austria - ELIN
Forschunzszentrum Gesellschaft m.b.H.
Ruthnergasse 1-7, A-1210 Wien
Austria

[Co-developer]

4.9.7 REFERENCES

[CCIT85a] Functional Specification and Description Language (SDL). Recommendations Z.100-Z.104 and Annexes, CCITT Red Book Vol. VI, Fasc.VI.10 and VI.11, Geneva, 1985.

[CCIT85b] CCITT High Level Language (CHILL). Recommendations Z.200, CCITT Red Book Vol.VI, Fasc.VI.12, Geneva, 1985.

[Theu86] N. Theuretzbacher, "HCDM: A Hierarchical Design Method for CHILL based Systems", Proceedings of 1986 International Zurich Seminar on Digital Communications, IEEE Catalog No. 86CH2277-2, pp.163-169.

4.10 HPS -- Hierarchical Planning for Software Development and Evolution

4.10.1 BACKGROUND

HPS models the software development process for long-life systems in terms of continuing evolution. Developed by C. Martin as an adaptation of MIT Professor A. Hax's Hierarchical Planning concept for manufacturing, this method was released for general use in December, 1988. It provides specific directions and procedures for project management activities, including resource planning and configuration management, throughout the software life-cycle.

4.10.2 DESCRIPTION

Hierarchical Planning for Software Development, HPS, is intended to help software development managers design an appropriate process for making resource allocation decisions. It makes the data processing or software development shop accountable for results. Through techniques which tie initial development to enhancement actions it is targeted for use in development of long-life systems.

The basic principles come from Arnaldo Hax's Hierarchical Planning approach for manufacturing. Its initial premise is that people at many different organizational levels make important planning decisions. The decisions at the highest levels cover the longest time period and largest breadth of systems work. Decisions made at one level should become natural constraints on decisions needed at the next level.

As adapted for software production, Hierarchical Planning starts with a special form of data flow diagram which describes the decision processes. It uses a six-level planning hierarchy that models the management of many complex software products and custom applications. The six levels are divided into strategic planning and release planning. Managers identify the nature of each type of resource decision, including the responsible individual or organization. Decision-making steps include overall budget-setting, scheduling releases, determining requirements, specifying the design, building the software, and requesting enhancements from operational experience. While providing a vehicle for management visibility and project control, the method pushes the decisions down to the lowest possible levels.

The method addresses problems of long-term consistency of requirements, design, and source code through baselining. It is designed to utilize data bases and specification documents produced by available CASE products to help design each new release based on what already exists.

The method looks at life-cycle issues from a different perspective than the waterfall and spiral software process models, but it is also compatible with these models. It is best-suited for use with an incremental or evolutionary development model. If 4GL rapid prototyping is used, HPS makes the organization decide how the rapid prototyping can be managed as an accountable effort.

4.10.3 TECHNICAL ASPECTS

The method's primary emphasis is support of MIS directors with long-life business applications. The developer states that is is well-suited for data processing or data base systems, expert systems or artificial intelligence and compatible for use on any applications which will go through evolutionary changes over a long period, such as 5-20 years.

HPS requires the use of data-flow analysis and stepwise refinement. Otherwise, it does not require specific programming practices, modes of representation, or analysis and design techniques.

As requirements evolve, the method points out explicitly how these changes relate to other resource planning decisions and assists in ensuring that consistency is maintained among specifications, design, and code. Decisions at each level become constraints on decisions at the next lower level. Changes at a lower level which affect higher level decisions must be discussed, approved, and documented at all levels involved. This is intended to foster accountability for results.

The method addresses reuse at the subsystem level, rather than for individual program units. The developer states that HPS leads to intelligent decisions when evolving the software about when subsystems should be rebuilt and when they should be carried forward with modifications.

4.10.4 PROJECT CONTROL AND COMMUNICATION

HPS shows how the various levels of a software development organization should plan their activities in order to be accountable and focus results on the users' needs. Although users are not part of the Hierarchical Planning process itself, user concerns are embedded in decision-making steps, such as determination of user requirements.

The method makes sure that documentation of each phase supports the decisions which must be made at that phase and others. It requires production of documents for requirements definition, functional specification, architectural specification, and design, as well as production of internal program documentation and user manuals, and use of a data dictionary, problem logs and change logs.

4.10.5 EASE OF USE

For successful use of the method, a software development manager or development team leader would require a bachelor's degree and three to five years of development experience on at least three or four different software systems, as well as an understanding of the classical life cycle view, i.e., of the basic activities of the software process.

An upper CASE or data base environment supports this method by maintaining up-to-date requirements and design specifications.

4.10.6 CONTACT INFORMATION

Charles F. Martin (508) 371-7011
Charles F. Martin Associates
P. O. Box 91
Concord, MA 01742 [developer]

4.10.7 REFERENCES

[Mart88a] C. F. Martin, "Hierarchical Planning for Large Systems", System Builder Magazine, Dec. 1988/Jan. 1989, pp. 46-53.

[Mart88b] C. F. Martin, "Hierarchical Planning for Evolution of Large Information Systems", proceedings of CASE'88 Workshop, Cambridge, MA, July, 1988.

4.11 InnovAda

4.11.1 BACKGROUND

The principal architects of this method are Richard Simonian and Michael Crone of Harris Corporation. Currently in development, the method is anticipated to be released for general use in 1989-90. The method's origins derive from object-oriented languages like Smalltalk and Flavors. InnovAda is intended to solve perceived problems such as Ada's lack of inheritance and dynamic binding, the need to port systems in Flavors to Ada, the need to develop hierarchies of objects, and the need to develop Ada-based, object-oriented knowledge bases and inferencing mechanisms.

4.11.2 DESCRIPTION

InnovAda provides enhancements, such as inheritance and dynamic binding, to the standard Ada language. It will allow a design to be prototyped on a LISP or Smalltalk machine, then to be ported to InnovAda. The result of the InnovAda preprocessor is compilable Ada code. The method can also be used as a stand-alone development environment.

The developer stated that the most effective use of the method is within the context of the rapid prototyping software process paradigm. The method is regarded as well-suited for the 4GL paradigm as well; the operational and transformational models were considered inappropriate for use with this method.

InnovAda is intended to be used just after requirements specification. The designer/user will choose some form of object-oriented design (e.g., entity-relationship, Buhr, or Booch diagrams) to isolate and refine the required objects, their inheritance hierarchy, and their interfaces. InnovAda does not explicitly support this step; the object-oriented analysis is assumed to be done off-line. Control analysis should also be performed to identify which objects can be implemented as InnovAda asynchronous objects.

The browser interface allows the user to remain in one environment while accessing InnovAda functions, the operating, system, editors, and the Ada compiler. The method uses terminology indigenous to object-oriented programming rather than terminology derived from procedural programming. Thus, objects are thought of in terms of class hierarchies, with (multiple) inheritance used to define new objects. Communication between objects is accomplished by means of message passing, in which the parameters included in the message are used by the receiving object to select the internal procedure (or "method") for executing.

The user interface provides templates for supporting InnovAda's object-oriented extensions. These templates allow the user to define new objects, define methods (procedures), generate a static object declaration that may be used as an initialization interface, and declare messages according to a consistent message-passing protocol. InnovAda either creates or modifies an Ada package specification and/or package body to implement the definitions of objects and methods. Modifications are made to the object or method templates, not to the generated Ada code. The developer stated that the Ada Code that is generated by InnovAda is machine-independent, commented, and suitable for delivery.

4.11.3 TECHNICAL ASPECTS

The method was reported to be well-suited for developing expert systems or artificial intelligence, and large scale simulation or modeling; it was considered inappropriate for developing embedded systems or process control applications, time-critical or real-time systems, and for systems programming. InnovAda is specifically intended to address artificial intelligence applications and is tied to the Ada programming language.

Modes of expression strongly encouraged by the method include entity-relationship diagrams, hierarchy charts, and Buhr and Booch diagrams. The method is seen as facilitating transformation across phases of the software process with its extensive code-generation facilities. Starting from high-level class and method templates, complete and compilable Ada source is generated.

Rapid prototyping is essential to the method for clarifying system requirements. Strongly-encouraged analysis techniques are data-structure, data-flow, and control-flow analyses.

The developer stated that once a base hierarchy of objects has been developed, new applications can be created by reusing and refining the existing objects. The loose coupling of InnovAda objects is said to assist efforts to reuse and modify these objects.

4.11.4 PROJECT CONTROL AND COMMUNICATION

InnovAda maintains its own libraries and projects; objects can also be traced with a CM system.

Clients can be involved at all phases of an incremental development since InnovAda's object-oriented approach encourages rapid prototyping.

Documents automatically generated based on data produced from other steps in the method are an interface specification and a system structure chart, both of which have a fixed format. Other forms of textual documentation are not currently addressed.

According to the developer, the method can be regarded as a specification-level language. It provides internal interface consistency checks to complement those made by the Ada compiler. The method assists in the early detection of inconsistencies and errors by internally tracking class and method definitions. Other inconsistencies or errors are detected by the Ada compiler.

4.11.5 EASE OF USE

The major theoretical constructs which, in the developer's opinion, should be understood by an experienced developer for successfully using the method include the object-oriented programming concepts of:

- multiple inheritance,
- information hiding,
- data abstraction,
- message passing,
- dynamic binding.

Additionally, knowledge of some object-oriented design methodology, including entity-relationship diagrams, is considered necessary.

The developer estimated the following as minimum qualifications needed by a development team leader for successful use of the method: a bachelor's degree, one to two years of development experience, knowledge of two programming languages, including Ada, and experience with three to four different software systems.

InnovAda currently runs under VAX/VMS. It requires the DEC Ada compiler. A subset of InnovAda (with no man-machine interface) will execute on any other Ada environment.

4.11.6 CONTACT INFORMATION

Richard Simonian 407-984-6006
Harris Corporation
P. O. Box 98000, MS W1/7736
Melbourne, FL 32902 [Co-developer]

4.11.7 REFERENCES

[Simo88a] R. Simonian and M. Crone, "InnovAda: An Object-Oriented Ada Enhancement", Presentations from the US Army ISEC Technology Strategies 1988 Conference, Alexandria, VA, Feb., 1988.

[Simo88b] R. Simonian and M. Crone, "InnovAda: True Object-Oriented Programming in Ada", The Journal of Object-Oriented Programming, (scheduled for winter 1988 publication).

4.12 MC -- Machine Charts

4.12.1 BACKGROUND

This method is an evolutionary development of ideas from Buhr diagrams [Buhr84] and CAEDE (Carleton Embedded System Design Environment); Raymond J. A. Buhr is the principal architect. The method provides a means of visual prototyping at the design level of behavior in the context of architectural structure. This focus on behavior and the consequent degree of language independence make the method useful in system as well as in software design.

4.12.2 DESCRIPTION

MC, Machine Charts, specifically addresses system specification, design, and implementation. It is founded on both object-oriented and event-oriented approaches. The emphasis of the method is on ways to reason about behavior rather than a fixed notation for specifying behavior.

The developer states that two types of modularity should be considered in the early design process: structural modularity and behavioral modularity. Structural modularity is associated with data and information hiding, for example, while behavioral modularity is associated with "agents of concurrency". or robots [Buhr88]. These robots are autonomous, self-directing machines. Separating them and their interaction protocols from the structural domain anticipates multiprocessing implementations, helps to localize the effects of timing changes in parts of the system, and reflects a natural way of describing the application problem. It is expressed by the developer that concern with these issues of macro-concurrency is appropriate in the design process and even the requirements process especially when dealing with multitasking or multiprocessing solutions for behavior-intensive applications.

MC provides iconographic components and connectors for its representation of behavior: "boxes" represent structrural partitioning, "robots" represent behavioral partitioning, and "buttons" and "fingers" represent interaction. Incorporated into these representation icons are semantics for conveying specific types of behavior. The iconographic notation is intended for use as a tool for thinking about behavior in the context of schematic diagrams using "scenarios". These scenarios are intended to help with reasoning about and explaning the behavior of key points in the system, which may then be described with other more complete and formal representations.

4.12.3 TECHNICAL ASPECTS

The method is not intended to address a specific type of application area, although the developer rated the method well-suited for embedded systems or process control, time-critical or real-time processing, and distributed processing or networks. The method is suitable for but not tied to Ada. Machine Charts separate the method from the language but are seen as providing better Ada support than either Buhr diagrams or CAEDE.

The "visual prototyping" paradigm of the method specifically provides a framework for reasoning about ano communicating problems and solutions of timing, concurrency, and fault tolerance by exposing those aspects of the system design contributing to these properties. The method emphasizes describing "how the system is supposed to work" in the context of how it is implemented architecturally.

Principal modes of representation are Machine Charts and Buhr diagrams. Several textual modes and other iconographical modes of expression are strongly encouraged. Mapping rules for translating from one mode to another are provided. These occur from requirements to concurrency domain machine charts, from scenarios to state machines, from total machine charts to code, and between concurrency domain machine charts and structure domain machine charts. The transformation from design to code is semi-automatic.

The method assists in reducing efforts needed to incorporate changes in the requirements by providing a framework at a higher level of abstraction than code in which the effects of the changes and the places in which they must be made can be understood. Consistency would ideally be maintained between levels of the system by constraining changes to the design level, through automatic regeneration of code; however, there is limited support for this in tools now.

Identification of possible reusable components is assisted by flagging such components in the charts and providing a way of drawing template diagrams separately.

4.12.4 PROJECT CONTROL AND COMMUNICATION

MC does not address activities usually associated with project management.

The software client is involved in the process in the sense that Machine Charts are used to explain design proposals to the client. Recent industrial experience has shown that clients can understand such charts.

Early detection of inconsistencies and/or errors is assisted by prototyping concurrent behavior separately and earlier than detailed data processing and computation. The method focuses early attention on aspects such as recovery from failure, correct sequencing of interactions, avoidance of deadlocks, critical races, starvation, and the like. Specific procedures are provided for generation of tests based on system requirements.

4.12.5 EASE OF USE

Concepts essential to the method include stepwise refinement, information hiding, process abstraction, abstract data-types, genericity, and module coupling and cohesion. Inheritance and use of assertions are compatible. The developer stated that finite state machines and concurrency concepts should also be understood for succcessful use of the method.

Two to three years of college-level technical education were considered minimum for successful use of the method by a development team leader. Development experience and knowledge of programming languages were considered unnecessary. The developer stressed that a non-sequential mindset and a systems viewpoint were required. He stated that hardware engineers with little software experience can use the method more effectively than data processing programmers with ten years' experience in five or more data processing languages.

In terms of automated facilities, various CASE companies have tools for workstations, e.g., Sun, Apollo, PC's, and MAC's.

4.12.6 CONTACT INFORMATION

Raymond J. A. Buhr, Professor 613-564-2735
Department of Systems and Computer Engineering
Carleton University
Colonel By Drive
Ottawa, Canada K1S 5B6 [Developer]

4.12.7 REFERENCES

[Buhr84] R. J. A. Buhr, *System Design with Ada*. Englewood Cliffs, NJ: Prentice Hall, 1984.

[Buhr85a] R. J. A. Buhr, G. M. Karam, and C. M. Woodside, "An Overview and Example of Application of CAEDE: A New, Experimental Design Environment for Ada", presented at International Ada Conference, Paris, France, May 1985.

[Buhr85b] R. J. A. Buhr, et al, "Experiments with Prolog Design Descriptions and Tools in CAEDE: an Iconic Design Environment for Multitasking, Embedded Systems", Dept. of Systems and Computer Engineering, Carleton Univ., Ottawa, Canada, Jan. 1985.

[Buhr88] R. J. A. Buhr, "Visual Prototyping in System Design", SCE Report 88-14, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, Sept. 14, 1988.

4.13 MMAIM -- Martin Marietta Ada Implementation Method

4.13.1 BACKGROUND

This method is in development and is anticipated to be released for general use in 1989. Its principal architect is Jeffrey R. Carter. The method is influenced by concepts found in OOD, JSD, Entity/Attribute-Relationship/Attribute approaches, PAMELA, and the work of Buhr.

4.13.2 DESCRIPTION

MMAIM, Martin Marietta Ada Implementation Method, deals with the software development process only, and specifically addresses system design and implementation. The following is an outline of the steps of the method.

1. Establish the software boundary and the interfaces through the boundary by creating the external entity graph (EEG).
2. Identify a reusable, fairly primitive software entity to handle the interfaces with each external entity. Identify the internal interfaces of these entities and their attributes. This is similar to Buhr's edges-in step.
3. Identify logical concepts in the problem statement which the software must handle. Create one software entity for each concept. Identify the interfaces of these entities and their attributes. Examine each entity for reusability.
4. Determine the interactions between entities. Steps 2-4 create the top-level entity interaction graph (EIG).
5. Mechanically generate Ada code from the EIG and compile it.
6. Repeat steps 3-5 for any non-primitive entities.
7. Identify behaviors and conditions for transitions between behaviors for all primitive entitites. This produces behavior transition graphs (BTG).
8. Mechanically generate Ada code from the BTG and compile it.

Step 6 and steps 7-8 may be performed concurrently. Each time code is produced, stubs may be added for incremental testing or for prototype development. Non-essential parts of the system may be stubbed for evolutionary or incremental development.

4.13.3 TECHNICAL ASPECTS

The developer rated the method well-suited for embedded systems or process control and time-critical or real-time applications; the method is intended to address these types of applications. The method is considered inappropriate for use with expert systems or artificial intelligence. Ada is tied to the method as an implementation language.

The method prescribes steps for handling concurrency and exceptions. The initial assumption is that all entities are concurrent. There is a notation for blocking and timeout considerations. There is also an emphasis on identifying and handling exceptions, and a notation for exception propagation.

The method requires structured English and a program design language as textual modes of expression. Icons specific to MMAIM are required for visual representation. These icons are intended to facilitate the transformation across phases of the software process by being mechanically translated into Ada code. The method prescribes mapping rules for translating from the entity interaction graph to Ada source code, and from the behavior transition graph to Ada source code.

Incremental or evolutionary development is essential to the method, and design reviews and code walk-throughs are strongly encouraged. Changes in the requirements are reflected in changes in the visualizations, from which the Ada code is mechanically generated. The fact that the code is generated from the visualizations is seen to ensure consistency between the design and code when changes are made to the visual representations.

Attention to reusability is emphasized early. Reusable entities are identified, allowing search for existing matching components; if the search fails, the component is designed and implemented for reuse. The developer stated that the notation is good for specifying reusable components.

4.13.4 PROJECT CONTROL AND COMMUNICATION

Project management activities are not addressed by MMAIM.

Quality assurance is addressed by means of the specific procedures for establishing early and enforcing interfaces, which is seen to facilitate the early detection of inconsistencies.

The method prescribes a fixed format for the behavioral, architectural, and interface specifications, as well as for the system structure chart and the data type dictionary.

4.13.5 EASE OF USE

Essential to the method are the concepts of information hiding and genericity. Other theoretical concepts which should be understood by an experienced developer for successful use of the method are finite state machines and state transition diagrams, abstract data types and abstract state machines; the developer felt an understanding of these concepts is useful but not mandatory.

It was estimated that a development team leader would need the following minimum qualifications for successful use of the method: two to three years of college-level technical education, three to five years of development experience, knowledge of one programming language, and experience with three to four different software systems.

There are no automated facilities as yet to support the method.

4.13.6 CONTACT INFORMATION

Jeffrey R. Carter 303-971-6817
Martin Marietta Astronautics Group
MS L0330
P.O. Box 179
Denver, CO 80201 [Developer]

4.13.7 REFERENCES

[Cart88a] J. R. Carter, "MMAIM - A Software Development Method for Ada; Part I - Description", SIGAda Ada Letters, Vol. 8, No. 3, May/June 1988, pp. 107-114.

[Cart88b] J. R. Carter, "MMAIM - A Software Development Method for Ada; Part II - Example", SIGAda Ada Letters, Sept./Oct. 1988.

4.14 PROSPECTRA -- PROgram development by SPECification and TRAnslation

4.14.1 BACKGROUND

The PROSPECTRA project aims to provide a rigorous methodology for developing correct Ada software and a comprehensive support system. Coordinated by B. Krieg-Bruckner, it is a cooperative project between Universitat Bremen, Universitat Dortmund, Universitat Passau, Universitat des Saarlandes (all Federal Republic of Germany), University of Strathclyde (Great Britain), SYSECA Logiciel (France), Dansk Datamatik Center (Denmark), and Alcatel Standard Electrica S.A. (Spain), and is sponsored by the Commission of the European Communities under the ESPRIT Programme.

Currently in development, with initial release anticipated in 1990, the method is available to European academic institutions for a nominal fee and may be made available to others upon inquiry. Although the method cannot as yet provide complete formal proofs of correctness that are sufficient to eliminate the need for final testing of large complex systems, it is anticipated that it will do so within the next few years.

The method is based upon the structured approach of the CIP group at Technical University of Munich, Federal Republic of Germany, with basic transformation rules of an axiomatic nature and upon the approach to algebraic specification methods provided by ANNA, ANNotated Ada. A related parallel project under the ESPRIT basic research action program is COMPASS, COMPrehensive Algebraic approach to System Specification and development.

4.14.2 DESCRIPTION

PROSPECTRA, PROgram development by SPECification and TRAnsformation, is an example of a formal method which follows the transformational model of the software process. Intended to make software development an engineering discipline, PROSPECTRA formalizes programming knowledge as transformation rules and techniques with the same rigor as engineering calculus. It provides program construction by application of these transformation rules and thus integrates verification with program construction during the development process.

Each transition from one program version to another can be regarded as a transformation in an abstract sense. In PROSPECTRA [KRIE88a], a transformation is defined as "a development step producing a new program version by application of an individual transformation rule, a compact transformation script, or, more generally, a transformation method invoking these. ... A transformation rule is a schema for an atomic development step that has been pre-conceived and is universally trusted, analogously to a theroem in mathematics. ... Transformations preserve correctness and therefore maintain a tighter and more formalised relationship to prior versions."

A simple development model of the major activities in the life of a program divides the life cycle into requirements analysis, development, and evolution. In requirements analysis, informal problem analysis and informal requirement specification are performed. The initial formal requirement specification is the starting point for PROSPECTRA. The user and the implementor agree on this formal specification as the interface or "contract".

With PROSPECTRA, a program is developed through a series of six language levels:

- formal requirement specification: loose equational or predicative specifications,
- formal design specification: specification of abstract implementation,
- applicative implementation: recursive functions,

- imperative implementation: variables, procedures, iteration by loops,
- flowchart implementation: labels, (conditional) jumps,
- machine-oriented implementation: machine operations, registers, storage as array of words.

The final version is obtained by stepwise application of transformation rules. These rules are carried out by the system, with interactive guidance by the implementor, or automatically by transformation tools. This is claimed to guarantee a priori correctness with respect to the original specification. That is, resulting programs are said to need no debugging, rather they are considered to be correct by construction according to the rigorous transformation rules. The only testing consists of validating the formal specification against the informal requirements, e.g., by prototyping.

In addition to formal specification and development by transformation, the method is founded upon a data structure-oriented approach and is compatible with object-oriented and control-oriented approaches. It may also be used with an incremental model and is compatible with the Waterfall model, the operational model, and 4GL models.

4.14.3 TECHNICAL ASPECTS

Intended to address safety-critical applications, the method is best-suited for scientific/engineering applications. PROSPECTRA is also compatible for use in the areas of embedded systems, process control, systems programming, distributed processing, data processing, and expert systems/artificial intelligence.

PROSPECTRA requires the use of ANNA, the formal specification language complement to the Ada implementation language, and encourages the use of mathematical notation. Hierarchy charts are required to show module decomposition and (re-)usage. No specific implementation language is conceptually required by the method, but Ada programs will be generated. The method facilitates the transformation across phases of the software process through built-in, extensible correctness-preserving transformation rules, scripts, and methods and support for program synthesis by completion techniques. In addition to requiring formal proof techniques,the method provides for proof of properties about the program, e.g., consistency of specification, hierarchical consistency, correctness of implementation/design.

PROSPECTRA manages complexity by abstraction, modularization and stepwise transformation. The algebraic approach is described in [Krie88b]. It uses loose algebraic specifications with partial functions and conditional equations to specify the properties of data and associated operations in PAnn dA-S, the PROSPECTRA Anna/Ada specification language.

Target constraints such as efficiency and other machine-oriented implementation detail are incorporated in the design by conscious decisions of the applications developer when applying the transformation rules. Although target constraints are not directly addressed by PROSPECTRA as yet, coupling this conceptual approach to Ada and ANNA allows utilization of Ada's facilities for concurrency constraints.

The method assists in reducing the effort needed to fully incorporate changes in the requirements through its logging of the development history, which can be replayed for changed requirements. It assists in ensuring that consistency is maintained among specification, design or code since transformation rules only allow correct development transitions. Proof may be required, assisted by the verifier. With this method, code is "a priori" correct with respect to the original requirement specification.

PROSPECTRA assists in identifying possible reusable components through its provisions for the components to be stored in a library, retrieval based on formal specification of an interface definition, and instantiation of generic components. A long-term goal is the potential for development of an industry of software components. Formal specification in Anna defines the semantics, while allowing the implementation to remain a company secret of the producer.

The choice of Ada/ANNA as a standard language and the use of standard tool interfaces as defined in the planned ESPRIT Portable Common Tool Environment (PCTE) is intended to ensure portability of the resulting application system, as well as of PROSPECTRA itself.

4.14.4 PROJECT CONTROL AND COMMUNICATION

The method requires the use of a tailorable format in conjunction with producing a requirements definition, functional specification, architectural specification, and interface specification. Automated support is provided to automatically generate system structure charts, and logs for design decisions, problems, changes, and program development history. The program development history log can be replayed. Version and configuration control are supported by the Library Manager.

The client is involved in the software development process in negotiation of the initial requirements specification, encouragement not to overspecify, and the use of abstract specifications, avoiding implementation/design detail.

Early detection and prevention of inconsistencies and/or errors is the main object of this method. This is accomplished through validation against informal requirements, proof of internal consistency of requirements specifications, and having the system control the application of correct transformation rules and methods. The stepwise construction process incorporates proof of correctness. Once the original specification is validated, no test of the final product is required, in theory.

4.14.5 EASE OF USE

As a minimum, a development team leader would require an advanced degree, three to five years of development experience, a working knowledge of two programming languages, and experience on one other software system. In order to successfully use the method, an experienced developer should have a firm mathematical basis to use formal methods and understand the algebraic specification of abstract data types, predicate logic, and term rewriting, which deals with efficient execution of operational specifications. In addition to these concepts, PROSPECTRA uses stepwise refinement, information hiding, process abstraction, abstract data-types, and genericity and is compatible with structured programming, inheritance, and the use of assertions for pre- and post-conditions.

The automated facilities provided by the method require a SUN 3/60 workstation, the Unix operating system and the Cornell Synthesizer Generator. Other useful software includes Prolog and ML, a functional programming language developed by Edinborough University and widely used in Europe. The Prolog provided by Quintus is more tightly integrated with the method, but other vendors' products may be used.

4.14.6 CONTACT INFORMATION

Professor Dr. Bernd Krieg-Bruckner 011-49-421-218-3660
PROSPECTRA Projekt
FB 3 Mathematik - Informatik
Universitat Bremen
Postfach 330440
D-2800 Bremen 33
Federal Republic of Germany

[Project coordinator
& principal architect]

4.14.7 REFERENCES

[Krie88a] B. Krieg-Bruckner, "The PROSPECTRA Methodology of Program Development, in Proceedings of IFIP/IFAC Working Conference on Hardware and Software for Real Time Process Control", Warsaw, 1988, North Holland publisher, pp. 257-271.

[Krie88b] B. Krieg-Bruckner, "Algebraic Formalisation of Program development by Transformation", in Proceedings of European Symposium on Programming, Nancy, March 1988, Springer-Verlag Lecture Notes in Computer Science No. 300, pp. 34-48.

4.15 REMORA

4.15.1 BACKGROUND

This method, currently in beta test, is concerned with large information systems and is targeted for general release in 1989. The developer states that the method is founded upon both event-oriented and object-oriented approaches.

4.15.2 DESCRIPTION

REMORA is a method which organizes the development of large information systems (IS) in two interdependent steps: a conceptual step and a physical step. The first step centers around the semantic representation of the real world system. This first step produces a solution in terms of a conceptual scheme. This conceptual scheme is a formal representation of the natural sturcture of facts perceived from both static and dynamic viewpoints.

The second step includes the technical aspects of the solution produced in step one. It complements the initial solution by introduction of parameters which were previously unnecessary.

Within both steps, the solution is obtained by modeling with theoretical concepts and tools and also represented in a formal language.

The conceptual step is supported by automated tools. These include a graphics interface for the IS conceptual schema specification and documentation, a prototyping tool, and an expert design tool which produces the IS conceptual schema starting with a set of French sentences describing the Universe of Discourse to be considered.

4.15.3 TECHNICAL ASPECTS

The method is intended to address applications specifically in the time-critical or real-time area, as well as data processing or database areas. It can be tied to any programming language using a manual mapping, and generates SQL embedded code.

The developer reports that timing constraints are specified and can be tested by prototyping; concurrency constraints are also supported.

The method requires a formal specification language for textual representation. Required iconographical modes are entity-relationship diagrams and dynamic graphs. There are rules prescribed for translating the graphical representation of dynamic sub-schema into a formal specification. These precise transformation or mapping rules are seen to facilitate the transformation across the phases of the software process.

In order to clarify system requirements or behavior, the method requires rapid prototyping, dynamic animation, and executable specifications. Required analysis and review techniques include data-structure analysis, formal proof techniques, and design reviews.

The developer states that the synthetic, global and precise view of the design product, clearly expressed both in a visual form and a formal specification form, is intended to assist in reducing the effort needed to fully incorporate changes in the requirements. There are formal checking rules to assist in maintaining consistency between components of the developing software as well.

4.15.4 PROJECT CONTROL AND COMMUNICATION

REMORA does not address project management activities.

The method provides for a conceptual step which typically involves the client in the definition of the system content.

The method prescribes a fixed format for the requirements, functional, and behavioral specification, as well as for the data dictionary. The functional and behavioral specificaitons are generated automatically based on data produced from other steps in the method. The interface specification has a tailorable format, and is produced from user responses to computer-directed prompts.

Consistency checks are defined at different levels of details at the different steps of the method. This is to assist in the early detection of inconsistencies.

4.15.5 EASE OF USE

The concepts of process abstraction, abstract data-types, gneericity, and use of assertions are essential to the method. Other theoretical constructs which should be understood by an experienced developer for successful use of the method include the concepts of object-event-operation and abstraction mechanisms like aggregation, generalization, and association.

The developer estimated that, for successful use of the method, a development team leader would need a *minimum* of an advanced degree of college-level technical education, no development experience, knowledge of one programming language, and experience with one software system.

There are computer-aided tools running on SUN workstations, written in C and Prolog.

4.15.6 CONTACT INFORMATION

University PARIS-1 46-36-97-61
12 Place du Pantheon
75231 PARIS Cedix 5
FRANCE [Developer]

4.15.7 REFERENCES

[Proi88] C. Proix, C. Cauvet, and C. Rolland, "An Expert System Approach for Information System Design", EDBT '88, Venice, 1988.

[Roll82] C. Rolland and C. Richard, "REMORA", IFIP Conference on Comparative Review of Information Systems Design Methodologies (CRIS), 1982, North-Holland, publishers.

[Roll88] C. Rolland et al., "The RUBIS System" in T. W. Olle, C. Bhabuta, and A. Verrign-Stuart, eds., CRIS 88 (Computerized Assistence During the Information Systems Life Cycle), North Holland, publishers.

4.16 TEDIUM

4.16.1 BACKGROUND

TEDIUM is a fully integrated environment supporting the development of software through the method known as System Sculpture. In the method, evolutionary development is used. The method follows the operational approach based on the use of executable specifications and product composition. Bruce Blum, the developer of TEDIUM, states that the system is currently under development. Versions of TEDIUM have been used for deliverable systems since 1980.

4.16.2 DESCRIPTION

The method of System Sculpture and the TEDIUM environment supporting it are designed for the development of information systems which have a high application risk, that is, where knowledge of what the application must do is poorly formulated. TEDIUM's goal is to manage an evolutionary development of the system.

Beginning with a preliminary set of concepts, the design concepts are specified through use of a formal language, and programs are automatically generated. It is then possible to experiment with the system in an interactive manner providing an opportunity for refining the interfaces, expanding the database, and modifying system functionality. Each change requires changing a specification and regenerating the program. Each regeneration produces a complete program. Once the behavior of a subset of the system is acceptable, the operational product is available for end-user experimentation.

The first step in the development process involves describing the system from three different viewpoints that define the behavior of the system:

- Requirements: what the application is intended to do;
- Data groups: what entities are to be modeled in the database;
- Processes: what functions and interactions are to be performed.

The designer then enters the above information into an application database through the use of TEDIUM. The database is a relational model with features that allow semantic information to be captured. Once the database models for the three viewpoints are defined, the program specifications can be defined and tested. Generic specifications are available for specialized, frequently used processes.

The developer states that the goal of the TEDIUM environment is to improve productivity by:

- Supporting the iterative development method of System Sculpture;
- Reducing the volume of source code in the form of expressive specifications;
- Formalizing and reusing standard programming conventions in the generic specifications and the program style;
- Reusing information to produce documentation.

The developer further states that TEDIUM satisfies criteria for an integrated environment which were proposed by L. Osterweil [Oste81]. These criteria specify that an environment should:

- Cover, for a given application class, all phases of the life cycle from initial requirements through maintenance;
- Support a unified user interface with on-line help tools;

- Be built upon an integrated application database that minimizes redundancy;
- Be created as a single application;
- Use the database for development and documentation.

In the opinion of the developer, TEDIUM differs from a method which uses fourth-generation language (4GL) for information system development. Generally, only parts of an application can be implemented using a fourth-generation language, and the use of such a 4GL requires additional resources. While 4GL's are intended mainly for end-users, TEDIUM is intended for use by developers. There are few built-in end-user tools, and there is no ad hoc query facility. Since the goal is to produce an efficient, custom application, the generator is designed to produce code that performs as well as hand-crafted code.

4.16.3 TECHNICAL ASPECTS

The developer reports that the method is well-suited for data processing or database applications, and is specifically intended to address interactive information systems. MUMPS is tied to the method as an implementation language, but this is not an essential limitation.

TEDIUM requires a formal specification language and strongly encourages specified documentation templates and narrative overviews of modules. All representations are in an integrated database and the system does the translation and merging.

Required techniques for clarification of system requirements or behavior include rapid prototyping, incremental or evolutionary development, and executable specifications. Facilitated Application Specification Techniques is a required review technique.

The method assists in reducing the effort needed to fully incorporate changes in the requirements by its compact representation, its program generation, and its integrated documentation. All design data is in a single integrated database, which assists in ensuring consistency in the developing software. Regarding assistance in identifying possible reusable components, the developer stated that the TEDIUM environment is built upon reuse: system style, generic programs, application code, and descriptive text.

4.16.4 PROJECT CONTROL AND COMMUNICATION

The developer described TEDIUM as supporting few project management activities.

The method's provisions for involving the client during the software development process are through interaction with the "prototypes", which the developer states are complete, e.g., all error tests have been done. In addition, the developer states that the "natural" definition of the product helps to involve the cuent.

The degree of tailorability of the documentation required to he produced varies from fixed format (data dictionary, design document, internal program documentation, user manual) to tailorable within the method (requirements definition, functional specification, behavioral specification, architectural specification, interface specification, and user manual) to unprescribed format (quality assurance/test plan, support/installation plan, design decision log, problem log, and change log). Those documents whose formats are prescribed are provided with automated support, either generated based on data produced from other steps in the method, or produced from user responses to computer-directed prompts.

The method provides specific procedures and automated support for prescriptive checking of interfaces. The developer states that the database will not accept many inconsistencies.

4.16.5 EASE OF USE

The major theoretical constructs which should be understood by an experienced developer for successful use of the method are data models. The developer gave the following as minimum qualifications needed by a development team leader for successful use of the method: less than two years college-level technical education, one to two years of development experience, knowledge of one programming language, and experience with one software system.

The hardware/software configurations needed to support the TEDIUM environment include PC up to VAX or PDP-11, UNIX machines, or IBM.

4.16.6 CONTACT INFORMATION

Bruce I. Blum — 301-953-6235
Johns Hopkins Applied Physics Laboratory
Laurel, MD 20707-6099 — [Developer]

4.16.7 REFERENCES

[Blum87a] B. I. Blum, "The Tedium Development Environment for Information Systems", IEEE Software, Vol. 4, March 1987, pp. 25-34.

[Blum87b] B. I. Blum, "An Overview of TEDIUM", Information and Software Technology, Vol. 29, No. 9, Nov. 1987, pp. 475-496.

[Oste81] L. J. Osterweil, "Software Engineering Research: Directions for the Next Five Years", Computer, April 1981, pp. 36-37.

An extensive bibliography is also available from the developer.

4.17 UOSE -- User Oriented Software Engineering

4.17.1 BACKGROUND

This method is in the exploratory research stage, targeted for release for general use in 1990. The principal architect is M. S. Deutsch of Hughes Aircraft Company. The method prescribes specific procedures for requirements definition or clarification, system specification, and verification and validation, and is based upon structured analysis, threads, and finite state machines.

The method particularly focuses on how to view the system in terms a user can understand. This concept entails delineating scenarios of external system behavior that are visible to a system user as the fundamental system view, a view that remains fundamental for validation throughout the system's lifetime. A distinguishing :haracteristic of this method is the specific integration of validation events with the traditional analysis and design activities.

4.17.2 DESCRIPTION

The User Oriented Software Engineering paradigm, UOSE, is an emerging method to define, implement, and validate complex, real-time systems by describing these systems from the viewpoints of the major parties in system development: the customer, the user, and the implementer. The models representing these points of view are, respectively, the requirements model, the operations-concept model, and the implementation model.

The modeling process is begun by developing an initial operations-concept model. A set of events associated with the devices in the external environment is keyed to this model. Then the events are modeled as stimulus/response patterns using a graphics tool called the system-verification diagram.

There are three steps associated with the development of this initial operations-concept model. First, a black-box definition is developed. Then one expresses the external events in the black box's format to define the stimulus/response elements. The elements are linked with the appropriate Boolean relationships to form a system-verification diagram that models the system's operations concept. Linking an external stimulus to an external response makes it apparent early on the need to express the basic response-time requirements that are important to the system user. The performance needs can be further allocated to internal functions as the system definition progresses.

After completing the initial operations-concept model, the initial requirements model is developed. However, the developer stated that there may be situations in which the development order does not progress in this way; in any case, the development of the two models is dynamically linked - one is refined based on adding detail to the other. A context schema is the first product of the requirements model. The context schema shows the embedded system as a single transformation connected to the dataflows and control-event flows that form the interfaces between the embedded system and the surrounding environment. The single transformation is expanded, in successive steps, into more detailed data-flow diagrams that show how the system behaves in response to external events as expressed in the operations-concept model.

When the initial versions of the two models are complete, validation step one occurs. This is accomplished by testing (on paper) the requirements model against each stimulus/response scenario in the operations-concept model; this validation is vital when separate parties develop the two models. Each scenario is traced through the data-flow diagrams; the detailed requirements text must also be correlated by paragraph number (or other identifier) with the scenarios. Any omissions, contradictions, or redundancies revealed in either model by this paper execution of the scenarios against the requirements may now be corrected.

When these two models have been reconciled, the implementation model is derived from the requirements model using whichever design practice is appropriate. UOSE only requires that the system's physical components (software and hardware) and their interconnections be identified.

The implementation model is then validated against the operations-concept model. The implementation model is tested (again, on paper) with the stimulus/response scenarios as test cases. One forms a thread of components, both software and hardware, that provides the external behavior demanded by each stimulus/response element. Each validated thread and its associated stimulus/response scenario is the initial step in planning system integration and the final acceptance test. Each thread can also be simulated to verify response-time requirements and develop them into an early prototype. Any discrepancies identified by this validation step are corrected.

The system is then incrementally implemented according to a calendarized plan designating a completion point for each thread. The physical modules (software or hardware) associated with each thread are built and tested proceeding according to the plan, thus incorporating a "build a little, test a little" philosophy. All implementation, testing, and integration are based on the threads, thus perpetuating the originally established user oriented scenarios across the entire development cycle.

4.17.3 TECHNICAL ASPECTS

The method is intended to address event driven systems, and is regarded as well-suited for embedded systems or process control, time-critical or real-time applications, and distributed processing or networks. UOSE is not tied to a specific language, but systems built using the method can best be implemented in languages with some object orientation such as Ada.

The basic unit of system definition, implementation, testing and integration is the stimulus/response thread. This provides a mechanism for specifying response time requirements. Furthermore, the stimulus/response behavior paths will highlight external interface requirements early. The stimulus/response paths also allow postulation of failure scenarios and encourage consideration of recovery or fault-tolerant responses.

Iconographical modes of expression which are required are finite-state diagrams, data-flow diagrams, control-flow diagrams, stimulus/response diagrams, and thread diagrams. The method prescribes mapping rules for translating from one mode to another. Specifically, mappings are prescribed from stimulus/response diagrams to finite state machine diagrams, from finite state machine diagrams to data/control flow diagrams, from stimulus/response diagrams and data/control flow diagrams to Ada packages, and from stimulus/response diagrams and physical design representation to thread diagrams.

Required analysis techniques are data-flow analysis, control-flow analysis, and thread analysis/validation. Regarding the method's assistance in incorporating changes in the requirements, changes are first made in the user oriented operational scenarios and then propagated into the detailed functional requirements. The method requires that the new set of requirements are validated by "executing" the requirements on paper against the operational scenarios. This is seen to assist in ensuring that consistency is maintained within the system.

4.17.4 PROJECT CONTROL AND COMMUNICATION

Project management activities are not directly supported by UOSE.

The method provides procedures for involving the client during the software development process by starting with the generation of stimulus/response scenarios depicting user visible behavior. These are depicted in simple diagrams keyed to being understood by non-technical participants, possible clients. All subsequent specification, design, and testing reference these scenarios, permitting client involvement throughout the end-to-end development cycle.

Documentation formats are tailorable within the method for requirements, functional, and behavioral specifications, as well as system structure chart, data dictionary, design document, and quality assurance or test plan. The method does not prescribe the format of the architectural specification or the internal program documentation.

Specific procedures are prescribed for test planning, generation of tests based on system requirements, unit/integration testing, acceptance testing, regression testing, and prescriptive checking of interfaces. The user oriented operational scenarios are regarded as the fundamental view of the system; all subsequent requirements and designs are validated against these operational scenarios.

4.17.5 EASE OF USE

Essential to the method are the concepts of stepwise refinement, information hiding, and process abstraction. Theoretical constructs which should be understood by an experienced developer in order to sucessfully use the method were given as stimulus/response paths, finite-state machines, data-flow diagrams, thread analysis for requirements/design validation and test planning, and object-oriented design.

The developer estimated that a development team leader would need, for successful use of the method, a minimum of a bachelor's degree, three to five years of development experience, knowledge of one programming language, and experience with three to four different software systems.

Support tool requirements for UOSE have not yet progressed to a point where a specific configuration has been identified.

4.17.6 CONTACT INFORMATION

M. S. Deutsch
Hughes Aircraft Company
Information Systems Division
P. O. Box 92919 - Bldg. S65/J301
Los Angeles, CA 90009 [Developer]

4.17.7 REFERENCES

[Deut88] M. S. Deutsch, "Focusing Real-Time Systems Analysis on User Operations", IEEE Software, Sept. 1988, pp. 39-50.

CHAPTER 5

OTHER METHODS CURRENTLY IN USE

This chapter contains descriptions of methods currently in use, adding to those presented in Chapter 3. These descriptions are based on information gathered for the first edition of this catalog and are presented here in abbreviated format.

The descriptions of methods in this chapter are based on the surveys used for the first edition; more current information was unavailable as of the publication deadline for the second edition. The developer survey used to collect information for the first edition was distinctly different from the developer surveys used in the current edition; therefore, semantic and temporal differences are assumed to exist for the information derived from the different surveys. Accordingly, to preserve information from the first edition but differentiate the source of such information, the authors have elected to provide the reader with summaries of these first edition methods in this separate chapter.

The reader should understand that each of the following descriptions is based upon data collected in 1987; this temporal "snapshot" of the method may differ from the representation of the method as it currently exists.

5.1 AUTO-G

5.1.1 OVERVIEW

Auto-G provides a computer-aided system which supports a method for functionally-oriented analysis and design of real-time systems. Auto-G addresses nearly all activities of the software process, with the only exception being project management. It requires an approach with emphasis on requirements and design as opposed to detailed coding.

The method was conceived within the International Telegraph and Telephone Consultative Committee (CCITT) as a formal superset of the facilities of the CCITT Specification and Design Language (SDL), the SADT method, and the Ada language. Its intent was to provide a more-powerful, yet graphically simplified representation compared to SDL.

5.1.2 DESCRIPTION

Auto-G supports a general method, DEMOS (DEcomposition Method for Object oriented Systems), which addresses analysis, specification and design of real-time embedded systems. DEMOS is based on REALM, a Real-world Abstract Logical Model, which views a running system as a set of asynchronously executing processes that communicate by the exchange of signals. Partitioning of a system into its constituent processes typically leads to a hierarchical system structure based on functional concurrency (required) and implementation concurrency (permitted).

Auto-G supports real-time systems development by:

- Providing a machine-checkable notation explicitly designed to model real-time functions and structures;
- Ensuring consistency and clarity at requirements capture, analysis, and specification;
- Increasing the visibility and, in turn, the correctness of design;
- Maintaining design visibility throughout successive stages of the design process;
- Eliminating the possibility of making syntax errors at the design stage and providing automatic logical consistency checking;
- Including configuration and version control as an intrinsic part of the design language.

The toolset associated with Auto-G provides for creation, storage, editing, and automatic analysis of documents. A "view" facility assists users in understanding a complex document by enabling them to look at consistent subsets of the contents. The same language, in either G (graphical) or T (textual) form, is able to express functional concepts at each stage from the incomplete viewpoint found in a requirement to the assignment statement found in detailed design. The use of G makes it possible to attach a formal meaning to each separate statement of functional requirement, and creates a formal machine-checkable statement of requirements. Existing designs may be reconfigured and reused to accommodate changes in the original requirements.

5.1.3 OTHER ASPECTS

Auto-G is well-suited for use in developing:

- Embedded systems, process control, or device control;
- Time critical or real-time applications;

- Distributed processing or networks.

Auto-G may also be used for developing:

- Scientific or engineering applications;
- Data processing or database systems;
- Systems programming;
- Expert systems or artificial intelligence;
- Image processing or pattern recognition.

The developer states that all of the concepts represented by data flow diagrams, finite-state diagrams, entity-relationship diagrams, Petri nets, Buhr diagrams, flowcharts, and other structure charts are incorporated in the G language, which is applicable from early concept exploration through detailed design. Auto-G uses hierarchical representations, narrative overviews, a formal specification language, and a program design language which is not based on a specific high-order language.

Auto-G implements the iconographical design language G, which is then used to provide G diagrams. The language is based on four main symbols, denoting a document, a function, an intermediate node, and a waiting node. Function is further augmented by the subsymbols for function type, basic actions, and basic objects. The relationships between symbols are indicated by connecting lines which represent action constructs or object constructs. The G notation has an isomorphic textual representation in the T notation. Any G diagram may be translated into an equivalent T design and vice versa.

Auto-G ensures consistency of specification, design, and code by requiring the use of the G or T language through all activities and in particular to generate all of the executable code. Thus, a backwards trace to the design is created inherently when the code is changed, and a backwards trace to the specification is created when the design is changed. Furthermore, the design may be entered or changed in either its graphical or textual form and updated versions of both forms are available.

Design involves decomposition into independent components with rigidly defined interfaces. Identification of likely changes is encouraged in design, in order to localize the impact of such future changes.

While some aspects of the method embodied in the Auto-G product could be used in the absence of the tool, the method cannot be separated from the automated tool without losing its essence. G and T are supported by a development support environment with a syntax driven graphical editor of G diagrams (Auto-G), a T language interface, a Semantic Checker for static consistency checking of the contents of G/T representations and a Functional Exerciser (Auto-X), for dynamic exercising and rapid prototyping of valid G/T configurations. A number of target code generator options are in development for direct generation of target code from G/T representations.

Training is available in the form of hands-on demonstrations, overview presentations, classroom tutorials, user manuals, an on-line help facility, 'hot line' service, users' support group, periodic technical updates, and on-site consulting by the vendor and by independent consultants.

5.1.4 CONTACT INFORMATION

Advanced System Architectures (ASA) +(276) 682756
Johnson House
73-79 Park Street
Camberley, Surrey UK [Provider]

5.1.5 REFERENCES

[CCIT86] International Telegraph and Telephone Consultative Committee, "G/T - A Natural Evolution from SDL", June 1986, pp. 1-26,A.1-A.38. (internal report from ASA Ltd, UK to CCITT).

[Hemd87] G. Hemdal and M. Reilly, "G Applied to the Requirements Capture and Functional Specification of a Domestic Heating System", March 1987. (panel problem for the Methodologies and Tools for Real-Time Systems Conference, March 2-3 1987, Tysons Corner, Virginia, USA).

5.2 DSSD -- Data Structured Systems Development

5.2.1 OVERVIEW

DSSD provides techniques for addressing almost all activities of the software development process. One of the approaches taken is to specify the outputs of the system first, and then work backwards into the design. The method is based upon structured techniques as well as on the separation of the logical design from the physical implementation.

DSSD is based upon the efforts of Ken Orr to adapt a design based upon the data structure of the problem, work pioneered by Jean-Dominique Warnier, to structured programming techniques. Also influential upon Orr's work were some concepts from IBM's HIPO (Hierarchy, plus Input, Process, Output). The method was first used for a deliverable system in 1977.

5.2.2 DESCRIPTION

Data Structured Systems Development (DSSD) follows nine phases for accomplishing systems development:

1. Project planning;
2. Requirements definition;
3. Design;
4. Building/testing;
5. Installation;
6. Operations;
7. Use;
8. Maintenance;
9. Evaluation and audit.

The outputs of each phase become inputs to subsequent phases.

In the requirements definition phase, both logical and physical requirements are defined. Logical requirements consider the application context, functions, and results. Physical requirements involve constraints, alternatives, ranking and selections. A critical step in this phase is the development of a set of output definitions, agreed upon by the software client, for the main functions in the system.

Entity diagrams, which show the entities involved and the transactions passing between them, are used as a tool for determining system scope and important relationships at the application level during the requirements definition phase. These entity diagrams are analyzed to identify the functional processes and decision support functions. Also required is the identification and definition of the results, or outputs, necessary to support the processes and define the logical output base. The software client is interviewed twice to obtain first general and then more detailed outputs expected, and an output sample is produced for validation. These outputs are entered into data dictionaries and then consolidated. The requirements phase ends with the creation of a system data dictionary containing information on user-specific and systemwide names, timing constraints between outputs, and interdependencies between outputs.

The design phase, like the requirements phase, is perceived as having both logical and physical components. The logical design phase is concerned with finding the minimum set of data structures from the outputs, considering frequency of data occurrence and file structures. This is followed by steps to define the data transactions and the process mappings for the data and data transactions.

5.2.3 OTHER ASPECTS

Particularly well-suited application areas for use with DSSD include:

- Data processing or database systems;
- Systems programming;
- Distributed processing or networks.

The developer stated that the following areas are compatible:

- Embedded systems, process control, or device control;
- Scientific or engineering applications;
- Time critical or real-time applications;
- Expert systems or artificial intelligence;
- Image processing or pattern recognition.

In early, middle and later design, hierarchical representations, entity diagrams, and Warnier/Orr diagrams are used to communicate levels of technical abstraction. Entity diagrams consist of things, or classes of things, that exist in the real world, and the transactions that occur between them. Warnier/Orr diagrams are hierarchical, left-to-right representations of the program structure. In addition to these modes, the method requires specified documentation templates, data structure analysis, design reviews and code walkthroughs. It encourages dynamic animation, simulation, or execution and is compatible with structured English, flowcharts, and entity-relationship diagrams.

The use of output-oriented design in the requirements definition phase is meant to ensure that the data produced by the system is a minimal and correct set. In addition, design reviews and walkthroughs are required and guidelines given for how to conduct such walkthroughs.

DSSD makes a clear distinction between logical design and physical design:

> "Logical design has to do with characteristics of a system that are present because of the application, despite the implementation. Physical design has to do with those characteristics that are unique to a specific design and result from various constraints."

The logical design is said to emphasize the user's problem rather than the specific computer upon which the application is built, thus enhancing portability.

Specific DSSD support tools offered by the provider include the following:

- The DesignMachine produces design deliverables from system requirements and design specifications recorded in the tool's data base.
- DocumentOrr is a documentation system which integrates graphics and text processing when run in conjunction with DesignAid by Nastec Corporation. It is meant to be used for both design and management documentation through the life of the project.
- Structures is a COBOL-based system for the design, construction, and maintenance of systems documentation, including Warnier/Orr diagrams. A code generator option for COBOL is also available.
- Brackets is an interactive Warnier/Orr diagramming tool which interfaces with Structures.

Training is available in the form of hands-on demonstrations, overview presentations and classroom tutorials, user manuals, a "hot-line" service, a users' support group, on-site consulting by the vendor, related publications from third-parties, and periodic technical updates.

5.2.4 CONTACT INFORMATION

Ken Orr and Associates, Inc.
1725 Gage Boulevard
Topeka, KS 66604

913-273-0653
800-KOA-8000
[Provider]

5.2.5 REFERENCES

[OrrK77] K. Orr, Structured Systems Development. New York: Yourdon Press, 1977.

[OrrK83] K. Orr, Structured Requirements Definition. Topeka, KS: Ken Orr and Associates, 1983.

[Higg87] D. A. Higgins, "Specifying Real-Time/Embedded Systems Using Feedback/Control Models", Proceedings of the Twelfth Structured Methods Conference, Chicago, IL, Aug. 3-6, 1987, pp. 124-147.

[Warn81] J. Warnier, Logical Construction of Systems. New York: Van Nostrand Reinhold, 1981.

[Warn74] J. Warnier, Logical Construction of Programs. New York: Van Nostrand Reinhold, 1974.

5.3 MERISE - the MERISE Method

5.3.1 OVERVIEW

The MERISE method uses two approaches to develop a software product. The two approaches are combined to provide tools for project management, design, and realization. The method encompasses nearly all aspects of the software development process from problem definition through detailed design. It also addresses issues of cost estimation, testing, and installation. The method is particularly well-suited to the development of information systems.

The method was developed by R. Colletti, H. Tardieu, and A. Rochfeld in cooperation with the French Ministry of Industry. The method was first used with respect to a deliverable system in 1979. The method is based upon concepts espoused in entity relationship models, in the use of CODASYL standards or Codd relational models for logical data representation, in the use of general system theory and Petri networks for process modeling, and in object-oriented design.

5.3.2 DESCRIPTION

The MERISE method involves the use of a double approach -- the level approach and the stage approach. Interaction between the two approaches occurs during a preliminary study which provides various solutions at the organizational and technical levels, during the detailed study of the chosen solution, and during the technical study which defines the technical specifications for the chosen solution.

The level approach begins with an abstraction cycle which involves the separation between data and processing. Within this approach, the method provides tools of formalism and rules for the construction of models. The problem is analyzed at three levels: a conceptual level, an organizational level, and an operational level. Two models are created at each level, a model for the data and a model for the processing which is to occur within the system.

At the conceptual level, management choices are taken into account through the Data Conceptual Model and the Processing Conceptual Model. At the organizational level, the interaction between man and machine is determined, and these organizational choices are reflected in the Data Logical Model and the Processing Organizational Model. At the operational level, technical choices are represented in the Data Physical Model and the Processing Operational Model. The link between data and processing is then made through two steps: 1) an internal validation which provides cross-checking between data and processing and which enables the detection of missing elements; and 2) an optimization which facilitates a decrease in the exploration costs of the information base by logical reduction of access and data storage.

The stage, or step-by-step, approach organizes the relationships and contacts between users and designers, specifying the user check points for each phase of the project, and providing external validations at each phase. Needed decisions are graded so that major decisions are taken at an early stage, and minor decisions postponed until later.

The stage level involves five major phases: the master plan, the preliminary feasibility study, the detailed studies, the realization, and the implementation. In the master plan phase, an analysis is made of the whole enterprise with the aim of defining its policy and strategy regarding the information system of the enterprise. During the preliminary feasibility phase, a study is made of the actual system, a model of a future system is developed, and various solutions are proposed along with the advantages, the disadvantages, and the costs for each of the solutions. This phase ends with the choice of one of the proposed solutions. The next phase involves the creation of an overall and a detailed description of the chosen solution. The phase concludes with the user's agreement on the general and detailed specifications. The realization phase is divided into two steps: 1) the technical study which involves developing a detailed description of the architecture, and 2) development of the software product. Finally, during the implementation phase, the system is installed and training in its use is provided.

5.3.3 OTHER ASPECTS

Use of the method is well-suited for development of:

- Data processing and database applications;
- Real-time or time-critical applications;
- Systems programming;
- Distributed processing or network applications.

The use of the method is compatible with the development of embedded systems or device control applications.

Systems have been developed for the banking industry, for the insurance industry, for the French national railways, for the Paris tube system, and for an electric power company.

The method requires the use of entity-relationship diagrams and a formal specification language from early concept exploration through to detailed design. Augmented Petri nets are used during concept exploration, while data-flow diagrams and narrative overviews of procedures are used during concept exploration and preliminary design. A program design language which is an extension of a high-order language can be used during the design phases. Mathematical notation, simulation, data structure analysis, control flow analysis, and design reviews are also used during development. Additionally, the use of specified document templates and other types of structure charts is encouraged.

The method ensures consistency of specification, design and code by requiring the use of a specification language. The effect of change is insulated by use of three abstraction levels. The method uses a complete modeling of the information system at an early stage in order to detect inconsistencies and/or errors. Analysis tools are used to confirm that the software product is complete before code generation is begun. A quality assurance plan is integrated into the entire process. The method incorporates tools within itself leading to verification, and also interfaces with other tools which specifically address verification issues.

Maintainability of the software product is supported by providing a detailed record of the design decisions. Portability of the software product is supported through use of the three abstraction levels, through use of standards and primitives for the target configuration, and by using tools which account for the parameter changes needed to generate various target configurations.

Automated tools which support the activities of the method include:

- DIAMANT: which supports construction of the logical data model and the physical data model, and which provides a data dictionary;
- METRADOC: which provides a processing model, an organizational model, and a processing operational model;
- a combination of the above two tools for use as a design generator.

All of the above facilities are also available in a complete integrated software tool call the METRA workbench.

Types of training assistance in the use of the method which are available include hands-on demonstrations, overview presentations, classroom tutorials, and on-site consulting by both the vendor and independent consultants. In addition, there is a users' support group, user manuals are available, and periodic technical updates are issued. Training courses are available for both the Level approach and for the Stage approach. These courses are given in both French and English.

5.3.4 CONTACT INFORMATION

SEMA.METRA 46 57 13 00
16 Rue Barbe's
92126 Montrouge Cedex, France [Provider]

5.3.5 REFERENCES

[Roch83] A. Rochfeld and H. Tardieu, "Merise: An Information System Design and Development Methodology", Information and Management 6, 1983, pp. 143-149.

[Roch86] A. Rochfeld, "An Information System Design and Development Methodology", Proceedings of the 5th International Conference on Entity Relationship Approach, held in Nov. 1986, to be published by North Holland.

5.4 NIAM -- Nijssen's Information Analysis Method

5.4.1 OVERVIEW

NIAM is a method for defining requirements and specifications for information systems. It uses natural language in order to better communicate with the end-user. Automated tools are available to support the method.

NIAM is based on concepts first elaborated by G. M. Nijssen. It is closely related to the method PRECISE. It was first used for a deliverable system in 1978. NIAM was one of the methods included in the METHODMAN I survey.

5.4.2 DESCRIPTION

Nijssen's Information Analysis Method (NIAM) addresses requirements definition and specification. It employs the concept of semantic information modeling, using graphical structure charts to model the system.

The developer states that the main characteristics of the method are:

- It has a strong user-orientation;
- It is based on natural language;
- It can be used for information structure analysis as well as for knowledge acquisition in building an expert system;
- It is completely independent of any constraints associated with computer technology.

Additionally, the method uses concepts and terminology which are incorporated into an international standard. [VanG82]

The method assumes that that accomplishing particular activities is dependent upon completion of prerequisite activities. The method is compatible with the use of functional and data hierarchical decomposition.

The developer emphasizes that it is necessary to abstract from any technical aspects, and to get end-users involved through the use of examples and population diagrams.

5.4.3 OTHER ASPECTS

Use of the method is well-suited for scientific or engineering applications, for data processing or database applications, and for the development of expert systems and other artificial intelligence applications. Use of the method is compatible with the development of embedded systems, process or device control systems, and time-critical or real-time systems.

The method uses a formal specification language, data-flow diagrams and other structure charts during concept exploration and preliminary design. An automated tool (IAST) is available which provides semantic and/or syntactic analysis for the formal specification language and for data structure analysis. The tool also supplies syntactic analysis for data-flow diagrams and other structure charts used by the method.

In order to assess the conformity of the developing software to system specifications, the method uses a formal specification language, and incorporates tools leading to verification.

In addition to the tool IAST which is supplied by Control Data, the tool QINT TINA, supplied by QINT Database Corporation, is recommended for use in supporting the activity of information structure analysis.

Asssistance available for training an organization in the use of the method includes overview presentations, classroom tutorials, on-site consulting by the vendor or by independent consultants, and related publications from third-party sources.

5.4.4 CONTACT INFORMATION

Contol Data Corporation 612-853-6628
511 11th Avenue South
Minneapolis, MN [Developer]

5.4.5 REFERENCES

[Verh82] G. M. A. Verhegen and J. Van Bekkum, "NIAM: An Information Analysis Method", Proceedings of the IFIP/TC8 Working Conference on a Comparative Review of Information Systems Design Methodologies, Noordwykerhout, Nethederlands, May 1982, North Holland Publishing Co.

[VanG82] J. J. Van Griethuysen, editor, "Concepts and Terminology for the Conceptual Schema and the Information Base", Report of ISO TC/97/SC5/WG3, March 1982.

5.5 PRECISE

5.5.1 OVERVIEW

This method addresses problem analysis, requirements definition, specification, and preliminary design. The method is primarily targeted to the development of information systems which have database management system support.

PRECISE has evolved from Nijssen's Information Analysis Method (NIAM). The automated mapping procedure that is used in the method was developed by F. Van Assche, D. Simons, and M. Vanhoedenaghe. The developer reports that it was first used for a deliverable system in 1976.

5.5.2 DESCRIPTION

The PRECISE method is primarily intended for the development of information systems. The method addresses problem analysis, requirements definition, specification, and preliminary design. The method provides a prescriptive technique for developing the database schema definition for the system.

A user oriented information model, called the binary conceptual schema, is constructed by using the binary relationship model. The main characteristics of the binary conceptual schema approach are:

- It is close to a natural language, making easier to involve end-users in the specification of the information model;
- It distinguishes between objects and identifiers for objects;
- It recognizes subtypes;
- It allows the specification of inference rules and integrity rules;
- It is formal and machine processable;
- The structure can be graphically represented using the information structure diagram techniques of G. Nijssen.

Prototyping and feedback sessions with the client are used to confirm the completeness and workability of the conceptual data model. An automated mapping is then specified which defines a fifth normal form database schema from the binary conceptual schema. This mapping is accomplished with the assistance of the Information Analysis Support Tools.

The mapping involves three stages:

- The binary conceptual schema analysis in which implications, inconsistencies, and redundancies are identified in the binary conceptual schema;
- The reference type evaluation and generation which checks to see that certain referenceability requirements are met;
- The grouping stage in which storage and manipulation efficiencies are addressed.

The developer states that use of sophisticated approaches such as the binary conceptual schema has made automatic mapping to the DBMS schema an absolute necessity. The automated system can perform a thorough analysis, and make accurate groupings. For very large information systems, a manual mapping is time consuming and likely to be inaccurate.

5.5.3 OTHER ASPECTS

The method is well-suited for data processing, database, expert systems, or artificial intelligence applications. The method is also compatible for developing scientific, engineering, real-time, distributed processing, or network applications.

The primary modes of expression used by PRECISE are the binary relationship model and hierarchical representations during concept exploration and early design. The method also requires the use of data-flow diagrams, narrative overviews of procedures, and other structure charts during concept exploration and specification. A formal specification language, data structure analysis, and design reviews are also used.

The developer states that if a change in requirements leads to a change in the data structure, then the method will reverify the data model, and a new schema for the DBMS will be automatically generated. The developer also states that tools are used to automatically generate a data definition for a different database management system, thus reducing the effort needed to port the end-product.

Tools are available which supply semantic and/or syntactic analysis for the binary relationship model, data structure analysis, and data-flow diagrams. The tools can also provide support for dynamic animation, simulation, or execution.

The following types of assistance are available for training: hands-on demonstrations, overview presentations and classroom tutorials, on-site consulting by the vendor and independent consultants, and related publications from third-parties. Additionally, there is a users' support group.

5.5.4 CONTACT INFORMATION

Control Data Corporation 613-598-0221
130 Albert Street
Ottawa, Canada KIP5G4 [provider]

5.5.5 REFERENCES

[Vana83] F. Van Assche, D. Simons, and M. Vanhoedenaghe, "The Automated Mapping from a Binary Conceptual Schema to a 5th NF Data Base Schema", published by the International Center for Information Analysis Services, Control Data Belgium Inc., 1983.

[Nijs79] G. M. Nijssen, F. Van Assche, and J. Snijders, "End-User Tools for Information Systems Requirements Definition", in Formal Models and Practical Tools for Information System Design. New York, NY: North Holland Publishing Company, 1979.

5.6 SAM -- Syslog Automation Methodology

5.6.i OVERVIEW

This commercially available automated development environment contains computer supported tools such as interactive editors, report generators, code generators, query facilities, on-line analyzers/verifiers and a relational data-base. Software systems are represented as networks of processes. Source-code is automatically generated from specifications after verification. This method is considered by the developer to be appropriate for the detailed design through to the testing, installation, and maintenance phases of the software process.

The method was first used with respect to a deliverable system in 1975. According to the developer, it expands upon the concept known as Strategy Oriented System developed by W. M. Jaworski and his associates at Concordia University, Montreal.

5.6.2 DESCRIPTION

The Syslog Automation Methodology (SAM) allows software systems to be represented as networks of "processes", where each is a "strategy" executing in an "environment". The strategy is thought of as control-flow while the environment is thought of as data-flow. Strategies, environments, and their components are represented relationally.

Activities addressed by the method include: problem definition, requirements definition, specification, preliminary design, prototyping, detailed design, coding, documentation, unit testing, verification, validation, and maintenance. The overall approach is characterized by the developer as being functional hierachy or decomposition.

The SAM database is structured to allow multiple views of the system so that verification and consistency checking is performed at the specification level. Automatic computer-based tools support verification. Source code is automatically generated from specifiations once verifcation is completed. A single representation format is used throughout the the design phases, thereby addressing conversion issues related to differing representations.

The developer states that SAM is particularly well suited to the development of large and complex scientific and industrial applications.

5.6.3 OTHER ASPECTS

While the developer stated that the method is well-suited for embedded systems or process control as well as for scientific or engineering applications, he added that the method is compatible with the following applications areas: data-processing or data-bases, time-critical or real-time applications, systems programming, expert systems or artificial intelligence, distributed processing or networks, and image processing or pattern recognition.

The developer emphasized that the method is inappropriate for small-scale projects, pointing to the excessive overhead that would be generated in such cases.

The following representations are used in early, middle and late phases of the design process for communicating multiple levels of technical abstraction: structure charts, action clusters, hierarchical representations, narrative overviews, and a program design language not based upon a specific high-order language.

Required to communicate the design are: structured English, a program design language, pictograms called "action clusters" and design reviews. Additionally, dynamic animation, simulation or execution is encouraged along with data structure analysis, control flow analysis and code walk-throughs.

The method enforces a relationship between specification and code. Changes in requirements need to be analyzed with respect to the specification. Design changes are traceable back to the specification and code changes are traceable back to the design. The method address reliability through the use of executable models and automatic tools for verification. Decision logic is verified for consistency and completeness. The developer stated that the abstract description of the system is independent of the target system, thereby reducing the effort required to port to new targets.

The developer stated that SAM is incorporated in an automated tool supporting other activities beyond the method and that the method can be incorporated into an integrated environment.

Automatic tools available with this method provide syntactic analyses for specified documentation templates, structured English, program design language, structure charts, action clusters and control flow.

The following is available for training in the method: hands-on demonstrations, overview presentations, classroom tutorials, user manuals, on-line help facilities, on-site consulting by the vendor, and periodic technical updates.

5.6.4 CONTACT INFORMATION

Charles N. Small — 514/340-9233
SYSLOG Inc.
4996 Place de la Savane
Montreal, Canada H4P 1Z8 — [Developer]

5.6.5 REFERENCES

[Jawo86] W. Jaworski, I. MacCuaig, T. Marinelli, and T. Nyisztor, "Executable Specification for a Large Industrial Process", Proceedings of the Canadian Conference on Industrial Computer Systems, (Montreal 1986), pp. 60-1 through 60-5, Canadian Industrial Computer Society, 1986.

5.7 SBP -- Specification Based Programming

5.7.1 OVERVIEW

This commercially available method addresses the design, specification and synthesis of software. Stepwise refinement is performed with assistance from tools which are designed to incorporate information, or knowledge, of the target system, the programming process and appropriate previous designs. A formal language, which includes the ability to make assertions, is an integral part of the method.

The method was first used in 1985 with respect to a deliverable system. References supplied by the method's vendor indicate that Steven Westfold is the developer of the method.

5.7.2 DESCRIPTION

Specification Based Programming (SBP) is a method which addresses the software process primarily within the requirements, specification and design areas. Abstract specifications may be communicated by means of a formal language called REFINE. Validation, modification and enhancement activities are performed at the level of abstract specifications. The aim is to free the developer from the burden of programming detail by placing this burden upon a knowledge-based support environment.

The vendor states that knowledge-based techniques are used to incorporate the details of programming and the target implementation. Part of this process includes the formal use of automated stepwise refinement. Successive refinement steps are made with the support of the development system. System requirements spelled out as pre- and post-conditions are continually checked as more detail is added. Thus, the refinement steps are correctness-preserving, with the requirements still being satisfied after refinement. The target implementation may be a programming language such as Ada.

The REFINE language is said to be a very-high-level language, similar to SETL [Schw75]. It includes constructs for expressing set-theoretic and first-order logic relationships.

The environment provided includes language-oriented components such as a parser, a printer, a compiler, a type analyzer and an editor. The development tools portion of the environment includes a command interpreter, a context mechanism, a documentation system and a knowledge-base browser. Additionally, rapid prototyping and simulation techniques are used to validate behavior.

5.7.3 OTHER ASPECTS

While the method is best suited to scientific or engineering applications as well as expert systems or artificial intelligence, the method is also compatible with the types of applications listed below:

- Data processing or database;
- Systems programming;
- Distributed processing or networks;
- Image processing or pattern recognition.

The vendor stated that the method is inappropriate for real-time embedded applications.

The REFINE language together with hierarchical representations including a graphics component is used to communicate the design. This may be done during early design activities, as well as in middle and later design stages. Using structured English, mathematical notation, data-flow diagrams and control-flow diagrams is also encouraged.

The specification language is used to generate executable code. This permits changed specifications to be installed in the system by recycling through the design steps related to the changed specification. A record of design decisions and personnel involved is logged in a mechanized way; a trace through the design decision process is available.

The assertion aspect of the REFINE language permits the use of pre- and post-conditions within the specification. These obligations can be checked for consistency and completeness in an automated way. Transformation from design to implementation is carried out in an automated way, using correctness-preserving rules.

Since the target code is generated automatically, all that is needed to port an end-product to a new configuration with the same specification is the appropriate information or knowledge about the new configuration within the code generator.

The use of a formalized design and specification language may support efforts to reuse components since the reuse of a component requires an accurate specification of its behavior. The method permits queries of a knowledge-base for such a purpose.

The vendor statcs that the tools supplied with the method provide semantic and syntactic analysis in the following modes of communication:

- Documentation templates;
- Structured English;
- Programming design language;
- Formal specification language;
- Mathematical notation;
- Iconographical notation including flowcharts, finite-state diagrams, entity-relationship diagrams, other structure charts and data-flow diagrams;
- Dynamic animation, simulation or execution;
- Data structure analysis;
- Control flow analysis.

The following are available in support of training: hands-on demonstrations, overview presentations, classroom tutorials, user manuals, on-line help facilities, a "hot-line" service, on-site consulting by the vendor,related third-party publications, and video tapes.

5.7.4 CONTACT INFORMATION

Note: The acronym SBP is not a standard product name.

Steve Katzman 415/494-6201
Reasoning Systems, Inc.
1801 Page Mill Road
Palo Alto, CA 94304 [Vendor representative]

5.7.5 REFERENCES

[Schw75] J. Schwartz, "On Programming: An Interim Report of the SETL Project", Technical Report, Courant Institute, New York University, New York, 1975.

[West87] S. J. Westfold, L. Z. Markosian, and W. A. Brew, "Knowledge-Based Software Development from Requirements to Code", unpublished report supplied by Reasoning Systems Inc., Palo Alto, CA.

5.8 SEPP -- Software Engineering Practices and Procedures

5.8.1 OVERVIEW

This method provides a framework for software engineering in accordance with DOD-STD-2167. SEPP addresses equally both the administration aspects of software engineering and the software development aspects. The components of the method consist of 17 high-level requirements documents, supported by approximately 100 procedures for meeting these requirements. The method allows for choosing various techniques and tools without prescribing specific ones.

SEPP was developed by the Software Engineering Division of Hughes. It was first used in 1986 on a deliverable project.

5.8.2 DESCRIPTION

SEPP, Software Engineering Practices and Procedures, is a company proprietary method available to the Government for tailoring to a contract. The method provides a framework for addressing software engineering activities within the development environment, according to the standards of DOD-STD-2167. As such, it includes guidelines for accomplishing the administrative as well as technical project tasks.

SEPP consists of 17 practices in two volumes which establish the requirements for what, who, and when. Book 1 addresses the business administration aspects of software engineering. Support is provided for many of the activities associated with management.

Book 2 addresses software development aspects, including planning, software quality, configuration management, design, simulation, coding, test planning, unit testing, integration, verification and validation, and documentation. Most of these activities are dependent upon specific prerequisite activities, and do not proceed until their prerequisites have been completed.

At a more detailed level are approximately 100 procedures which establish the standards for all of the products required by the 17 practices.

SEPP considers software engineering aspects at a high level of abstraction, and therefore does not advocate specific design techniques and tools. Guidelines are provided, however, for addressing problems at a lower level of detail, i.e., the constraints imposed upon the target system, tailoring the method to specific features of the target architecture, target operating system and implementation language. These guidelines are particularly geared to applications involving embedded systems, process or device control, time critical or real-time constraints, and distributed processing or networking. Portability and reusability aspects of the target software are specified in documents concerning software quality requirements.

5.8.3 OTHER ASPECTS

The method was reported to be well-suited for the following application areas:

- Embedded systems, process control, or device control;
- Time critical or real-time applications;
- Distributed processing or networks.

The developer judged the method to be inappropriate for:

- Systems programming;
- Expert systems or artificial intelligence;
- Image processing or pattern recognition.

Because SEPP is a framework for software engineering, it does not prescribe any specific design technique, other than requiring specified documentation templates.

The method uses formal configuration control to assist in making changes to the system. The rationales supporting design decisions are recorded as the software evolves, as well as in later stages of design modification. The method addresses consistency by ensuring that code evolves from specifications, and that a backwards trace from code to design or from design to specification is created when changes are made.

The developer regarded as inappropriate the use of SEPP when programming small-sized applications.

Besides the configuration system (CCARS), there are other automated tools which support various activities of SEPP. These include HASEE, AIDES, and AIM , design tools provided by Hughes, as well as PSL/PSA for requirements analysis. The developer stated that there were approximately 50 different tools that could be used with this method.

Assistance available for learning SEP? includes overview presentations, classroom tutorials, user manuals, "hot line" service, on-site consulting by independent consultants, and periodic technical updates.

5.8.4 CONTACT INFORMATION

Software Engineering Division 714-732-1488
Hughes, Ground Systems Group
Box 3310
Fullerton, CA 92634 [Developer]

5.8.5 REFERENCES

See DOD-STD-2167.

5.9 SSPM -- Software Standards and Procedures Manual

5.9.1 OVERVIEW

This method is based upon a hybrid approach combining functional decomposition and object-oriented design in the implementation of DOD-STD-2167. It provides guidelines for analyzing the requirements and deriving the preliminary and detailed designs, with reusability as a major objective. Many activities in the technical and administrative areas of the development process are addressed. The method is company proprietary and available to the Government.

This method was developed by the Ada Technology Support Laboratory at Lockheed. As yet no deliverable systems are associated with the method. The method is based upon Yourdon's Structured Analysis for Real-Time Systems, and incorporates ideas from functional decomposition and object-oriented design.

5.9.2 DESCRIPTION

SSPM, "Software Standards and Procedures Manual", sets forth Lockheed's software engineering practices. Nearly all activities of the software development process are addressed. The method uses a combination of approaches in sequence, and follows the life-cycle paradigm embodied in DOD-STD-2167.

The method contains a strategy for deriving a requirements specification that serves as the basis for further preliminary and detailed design stages. This strategy consists of several steps: 1) The given, or inherited, requirements of the system are analyzed for consistency in order to improve nomenclature and introduce sequence; 2) Physical references in the revised requirements of Step 1 are rendered out; 3) The non-physical requirements are fully distributed; 4) Knowledge of the consumer is removed from the distributed requirements; 5) Commonality of the requirements is factored out, resulting in generic pruned requirements; 6) A functional Advanced Design Language (ADL) is produced. The internal reusability resulting from Steps 4 and 5 is recorded in a savings document.

At the preliminary design stage the major steps involve: 1) Decomposition of the requirements into detailed requirements; 2) Production of a declarative ADL; 3) Refinement of the requirements, producing a top level CSC design; and 4) Refinement of the declarative ADL.

The developer states that the transformation from preliminary to detailed design is the most critical point for reuse. At this stage the data types and their attributes are identified, as well as the operations on the data types and their module structure, interfaces between modules from N-squared charts are established, and the declarative ADL is refined to document the interfaces. The processes derived from functional decomposition are transformed into an object-oriented design. The key to this transformation is that each process may act upon only the next or the same level of object abstraction. Processes are grouped according to the objects upon which they act.

An objective of SSPM is to coordinate communication between the technical, management, and software client segments of the project. Information pertinent to managers is incorporated at all stages of the technical development effort.

5.9.3 OTHER ASPECTS

The method was reported to be well-suited for the following application areas:

- Embedded systems, process control, or device control;
- Time critical or real-time applications;
- Distributed processing or networks;
- Image processing or pattern recognition;
- Message processing.

Modes of expression which span all stages of development include hierarchical representations, a formal specification language, and an Ada PDL. In early stages of development, data-flow diagrams, system diagrams, and hierarchy charts are used. These modes are supplemented in middle stages with narrative overviews of procedures, subroutines, and packages. Later stages of development are characterized by other structure charts and narrative overviews.

All of the above modes of expression are required by the method, as are specified documentation templates, design reviews, and code walkthroughs.

The method incorporates a configuration management scheme tied to a specific tool. The method addresses consistency by ensuring that code evolves from specifications and from the use of a high-level PDL, and that a backwards trace from code to design or from design to specification is created when changes are made.

Early detection of inconsistencies is facilitated by the use of a high-level PDL, as well as by Data-flow diagram (DFD) consistency checking. Developers' checklists and system development folders are used to confirm completion of procedures, and requirements traceability is used to confirm the completeness of the software product.

Reusability of the software product is designed into the method by means of Ada, reuse documents, and a software library.

This method is incorporated in an automated tool which supports other activities besides those of the method. A configuration management scheme is included in the method and parameters concerning management information can be exported to external tools. The method can be incorporated into an integrated environment as well.

Assistance available for learning this method includes hands-on demonstrations, overview presentations, classroom tutorials, user manuals, and "hot line" service. The method is available to the Government; otherwise, it is company proprietary.

5.9.4 CONTACT INFORMATION

Note: "SSPM" is the vehicle which incorporates the method.

Ada Technology Support Laboratory 408-756-7774
Lockheed Missiles and Space Co.
Orgn 62-81 Bldg 563
P. O. Box 3504
Sunnyvale, CA 94089-3504 [Developer]

5.9.5 REFERENCES

[Cohe86] P. F. Cohen, "The Transformation of Functional Decompositions to Object-Oriented Designs", presentation at Ada Expo '86, Charleston, WV, 1986.

[Kapl86] J. S. Kaplan, "Using Ada Design Language to Achieve Internal Reuse", presentation at Ada Expo '86, Charleston, WV, 1986.

Lockheed, "Software Standards and Procedures Manual" (SSPM).

5.10 S-JAD -- Structured Joint Application Development

5.10.1 OVERVIEW

Structured-JAD is an accelerated analysis technique belonging to the FAST (Facilitated Application Specification Techniques) family. It brings together data processing professionals and end-users in an organized and formal meeting for anywhere from a few hours up to two weeks, in order to reach agreement on system specifications or resolve open development issues in a compressed time frame. It is not an analytical method, but rather a set of tools, techniques and procedures to be used to enhance the conduct of a system development method. Structured-JAD is based on the use of structured techniques. Information on IBM JAD is also provided in this writeup.

Structured-JAD is based upon Joint Application Development (JAD) from IBM, and Structured Analysis from DeMarco, Yourdon, and Gane/Sarson. JAD was developed by IBM in Canada in 1977. Four major FAST techniques have been described in [Rush85]: JAD, Consensus, Wisdm, and The Method. JAD from IBM focusses on the detailed external business design problem. Consensus was started by Boeing Computer Services and primarily addresses high level planning issues. Wisdm was started at Boeing Computer Services by Blair Burner, who formed Wise to market it. It also primarily addresses front-end analysis. The Method, from Performance Resources, modifies JAD for use with decision support systems. In addition to those four, Al Case describes Team System Analysis [Case87] and John Palmer utilizes "blitzing" [Palm87] in conjunction with Essential Systems Analysis.

5.10.2 DESCRIPTION

Structured-Joint Application Development (S-JAD) is designed to be integrated into an organization's existing system development environment. It is intended to reduce the elapsed time for acquiring a requirements definition and/or a system specification. It may use prototyping and may be used to produce the preliminary design as well. IBM's JAD also addresses risk analysis. Integration of the organization's standards into the Structured-JAD process enables production of the agreed-upon documentation as part of the meeting or immediately afterwards.

Systems analysis (the process of translating business needs to a system specification) is composed of three distinct sets of activities:

- Requirements definition, the capturing of user requirements for the proposed system;
- Alternative evalution, the analysis of different high level design alternatives generally conducted when the purchase of software or hardware products is being considered;
- External design, the design of an automated system to meet the user requirements from the user's point of view.

Structured-JAD sessions can be used to accelerate the completion of any or all of those activities. They are:

- formalized, with an agenda, purpose, and objectives established for a particular project;
- facilitated by a trained session leader;
- conducted in an interactive workshop environment;
- attended by both technical data processing specialists and end-users;
- documented in a pre-determined manner, consistent with the project type, organizational standards and available automated support tools;
- based on well documented usage of the structured techniques.

Use of Structured-JAD is dependent upon project requirements and facilities requirements. The project requirements include end-user support for the project, willingness of end-users and data processing personnel to spend 3-10 days in uninterrupted meetings, and a manageable size project (or the ability to partition it into smaller manageable pieces). Requirements also include an established project scope with a well-understood underlying business policy, and the cooperation of the project manager or leader. Facilities requirements include ample work space for the participants, suitability for long duration "roll up your sleeves" workshop environment, remoteness from the daily working environment of the participants, and availability of white boards, flip charts, screen(s) and overhead projector(s). Automated support such as CASE tools, that provide graphic models, text, forms, and a data dictionary, as well as prototyping, word processing, and project management tools are also advised.

5.10.3 OTHER ASPECTS

The method is reported to be well-suited for data processing or database systems, and expert systems or artificial intelligence; it is reported to be compatible with embedded systems, process control, or device control,scientific or engineering applications, and time critical or real-time applications.

S-JAD uses the techniques of structured analysis, such as finite-state diagrams, entity-relationship diagrams, data-flow diagrams, and narrative overviews for specification and preliminary design. It requires the use of specified documentation templates, e.g., specific forms designed for use in the meetings, as well as data structure analysis, control flow analysis, and design reviews. It encourages the use of finite-state diagrams and dynamic execution and is compatible with the use of Structured English, flowcharts, Buhr diagrams, and other structure charts. IBM's JAD uses flowcharts, other structure charts, hierarchical representations, and narrative overviews of procedures, subroutines, and packages.

Specific documentation is required for use in each type of session. For example, a project charter is used in requirements definition and contains project scope, assumptions, restrictions, constraints, any feasibility studies, and applicable planning documents. Alternative evaluation sessions require documented requirements and all pertinent vendor information for candidate software. External design sessions use documented requirements and details of a chosen implementation. Integration of the organization's standards into the Structured-JAD process is stated to help to increase quality and productivity. Documentation can then be produced as part of the meeting or immediately afterwards to avoid any after-the-fact transcription.

The method provides for prescriptive checking of interfaces, incorporates an executable model, provides for data-flow and control-flow analysis of dependencies, and incorporates the use of tools that address verification issues. The use of the joint meeting approach and the structured techniques is designed to ensure complete, consistant analysis and early detection of problems. Participation of both user and technical constituencies at JAD sessions is regarded as helping to ensure a complete specification. There is a review process built into the JAD session that covers both the deliverables and the development process.

Both a logical data model and process model are developed. The method uses structured analysis which partitions the system into loosely connected pieces; this partitioning is intended to facilitate reuse and maintenance. The implementation-independent models created with the structured techniques help designers port the system to multiple environments. The developer states that it would be inappropriate to use S-JAD on hardware and software conversion efforts.

The developer states that whether implemented on personal computers or through terminals tied to a mainframe, available automated support can and should be utilized in the Structured-JAD sessions. Specialized tools, such as CASE, will often require an additional session recorder for operation.

Training for S-JAD is available in the form of hands-on demonstrations, overview presentations, classroom tutorials, and on-site consulting by the developer and by independent consultants.

Training and information on JAD is available in the form of hands-on demonstrations, overview presentations, classroom tutorials, user manuals, on-line tutorials, on-line help, on-site consulting by independent consultants and video tapes.

5.10.4 CONTACT INFORMATION

Susan Ball 619-696-6644
Integrated System Solutions, Inc.
600 B street, Suite 1142
San Diego, CA 92101 [consultant, provider of S-JAD]

Sandy A. Montclare 914-578-3535
IBM Corporation
P.O. Box 700
Suffern, NY 10901 [contact for JAD]

5.10.5 REFERENCES

[ISSI87] Integrated System Solutions, Inc., "Structured-JAD", internal publication, 1987.

[Rush85] G. Rush, "A Fast Way to Define System Requirements", Computerworld, Oct. 7, 1985, pp. ID/11-16.

5.11 VDM -- Vienna Development Method

5.11.1 OVERVIEW

This method is a formal, or mathematical, method for developing software systems consisting of an orderly plan for the usage of specific tools, techniques and notational systems. Design rules show how to prove that a design satisfies its specification. The areas of the software process addressed include abstract specification and design; concrete design; the transformation and refinement of specifications into designs, and the proof of properties of specifications, designs and developed products. The method is prominent in Europe.

The Vienna laboratory of the International Business Machines Corporation originated the method in the 1970's. It developed from a related earlier effort known as the Vienna Development Language. VDM builds upon the work of H. Bekic, W. Henhapl, C. Jones, P. Lucas and D. Bjorner [Bjor87]. At present, D. Bjorner at the Dansk Datamatik Center is a principal developer. Additionally, C. B. Jones at the University of Manchester is very active in VDM development. Results of significance attributable to the method include the formal definition of PL/I, an implementation of an Ada language compiler and a formalization of the CHILL language.

5.11.2 DESCRIPTION

The Vienna Development Method (VDM) is intended to handle functional complexity in non-trivial systems. It is primarily a formal method for working out requirements, developing the detailed design and the code and for proving that the code has specified properties. The first requirement of the method is that an initial formal model of the system be constructed. This model is actually a specification. The formal model is step-wise refined with the result of finally obtaining executable programs. The formal specification and design language Meta-IV and a collection of reification techniques are used together to perform the elaboration and refinment. Each step in the development can be rigorously argued or proven correct.

The method teaches how to appropriately substitute rigorous argument for mathematical proof. This is done to address the assertion that proofs of correctness may demand too much mathematical formality to be practical. The developer stated that formal methods in general need further acceptance at the managerial and technical levels.

Mathematical abstractions are used to express data structures rather than constructs from a specific programming language. Each step of the design process records assumptions related to subsequent specifications about sub-components. Abstract representations of data are converted (data reification) to more concrete data types and these transitions are justified. The operations to be performed upon data are recorded at first in terms of pre- and post-conditions. Operational decomposition, the process of choosing, and justifying, a sequence of transformations which can be expressed in the implementation language, is then used to arrive at code which has been shown to meet specifications before its use.

The formalisms of the method may require more extensive mathematical training of developers than is expected when using other methods. Catalog readers are cautioned that "proof" or absolute certainty of correctness is unattainable in both physical and software systems; however, the method makes a significant effort to address correctness issues. Showing that informal user requirements are met may require testing, since there is no formal way to prove that such requirements are satisfied.

5.11.3 OTHER ASPECTS

The primary use is for systems programming. The developer states that it is well-suited for embedded systems, scientific/engineering systems, distributed processing and pattern recognition and processing. It is also stated that the method is compatible with data processing, real-time or artificial intelligence applications.

Completed projects include compilers for the programming languages: Pascal, Pascal Plus, OCCAM, CHILL and Ada as well as an interpreter for Prolog. The method has been used to produce specifications for other systems such as: IMS, System/R, System/2000, CODASYL DBTG, Pascal/R and database management systems. The developer stated that it would be inappropriate to use VDM for developing data-base and graphical man-machine interfaces that could be generated through the use of 4th-generation tools.

VDM incorporates a formal specification language known as Meta-IV; it uses mathematical notation. The use of structured English is encouraged.

The mathematical-formal techniques within the method provide a means for expressing proof-obligations, which are assertions that the code meets specifications. Rules for proving the assertions are part of the method. There are tools supporting verification.

Portability considerations are possible insofar as the target independent properties of a system can be specified. While reusability considerations are not specifically addressed by the method, the formal specification of an object's assumptions and properties is one of the most important ways to understand the capabilities and restrictions of a previously developed object.

The method can be incorporated into an integrated environment. There is a facility known as the Meta-IV tool-set; its specific capabilities were not obtained during the data gathering effort.

Overview presentations, classroom tutorials and on-site consultation are available from the vendor.

5.11.4 CONTACT INFORMATION

Dansk Datamatik Center ++45-2 87 26 22
Lundtoftevej 1 C
DK-2800 Lyngby
Denmark [Provider]

5.11.5 REFERENCES

[Bjor87] D. Bjorner, "An Annotated VDM Bibliography", published by the Dansk Datamatik Center, Lyngby, Denmark, 1987.

[Jone86] C. B. Jones, Systematic Software Development using VDM. Prentice-Hall, 1986.

[Luca87] P. Lucas, "VDM: Origins, Hopes, and Achievements", Proceedings, VDM-Europe Symposium '87, Lecture Notes in Computer Science, Springer Verlag, 1987.

This page is intentionally blank.

CHAPTER 6

SUMMARIES OF RESPONSES TO SELECTED SURVEY QUESTIONS

This chapter provides in tabular format a summary of the responses to selected questions from the Survey of Software Method Developers. The survey included questions intended to provide a basis by which methods can be compared. For such questions, the choice of responses was specified in the survey vehicle, thus providing a ready format for categorizing the responses. The questions selected represent suitable areas in which methods can be contrasted, and for which it is desirable to provide a basis of comparison. Additionally, the nature of the question should lend itself to determining the choice of responses. Typical of such questions are the application domains for which the method is suitable, the modes of expression used by the method, and the type of training that is available with the method.

It is recognized that restricting the choice of responses may limit at times the ability of the developer to describe fully some particular feature of a method. The use of free format response, however, does not readily provide a basis for data reduction. The free format response was used in Methodman I, and the researchers commented that, although the responses provided much detail, "the loosely structured answers made the job of reporting the results more difficult" [Free82]. This catalog provides an opportunity for developers to elaborate on specific features of the method, and reports the responses in the individual write-ups of the methods.

As part of the task of creating this second edition of the catalog, a self-assessment was performed of the survey efforts undertaken for the initial Software Methodology Catalog. As is typical of software development itself, much is learned from an initial effort. Based on this assessment, the survey questionnaire underwent an extensive revision, including the addition and deletion of questions, rewording of questions, and changes in the choices of answers provided for the objective questions. A copy of the survey questionnaire is provided in Appendix B. As a result of this revision, nineteen tables are provided in this edition, whereas the first edition contained only twelve tables. Furthermore, the authors believe that the tables included here contain more specific information than was provided in the initial catalog.

The methods which appear in this summary are those for which individual write-ups are given in Chapter 3. These methods represent ones currently in use, and are methods for which a response to the revised survey was provided by the developer. Since these tables are based on the revised survey, methods which appeared in the first edition of the catalog but for which a revised survey was not returned were not included in these summaries.

For certain questions, such as the question related to support for project management, some developers judged that the question was no\ applicable to the method, and no response was given. In order to provide the same basic format for all tables, a method was included even when no response was given for the particular question. Thus, the reader will find a method located in the same row position for all tables. Since the number of methods required the use of two pages for a single table, blank pages have been inserted in this chapter to ensure that the two parts of the table appear on facing pages in order to facilitate comparisons.

The reader is reminded that the source of information for this catalog is the developer or vendor of the method. It is natural to expect a bias in favor of the method, but at the same time one must recognize that the developer is the one who is best qualified to describe the features and intent of a method. Accordingly, in preparing this summary, the responses of the developer are reported as given. No attempt has been made to establish an independent validation of the responses, or to resolve possible inconsistencies that may exist occasionally in the responses of the developer to different questions. Although consideration was given to establishing some form of independent validation of the information provided, such an effort was judged to require resources beyond those available for the creation of this catalog. By in large, the information provided in this chapter is considered a fair representation of the methods listed.

It is hoped that the format used in this chapter will enable the reader to gain some insight into the similarities and differences of the various methods, and that such insight will provide a basis upon which to judge how well a particular method might fit one's needs.

Methods have been identified in the tables by acronyms. The following is an alphabetical list of the acronyms used and the corresponding full names for the methods.

METHOD ACRONYMS

ABDLSLCM	Ada Based Design Language System Life Cycle Methodology
ADM	Ada Development Method
AISLE	Ada Integrated Software Lifecycle Environment
BOX STRUCTURES	The Box Structure Methodology for Information Systems Development
BYRON	Byron PDL and Document Generator
CORE	Controlled Requirements Expression
DARTS	Design Approach for Real-Time Systems
DBO	Design by Objectives
DCDS	Distributed Computing Design System
DSSAD	Data Structured Systems Analysis and Design
E-DEV	Essential Systems Development
ESA	Essential Systems Analysis
GYPSY	Gypsy Methodology
HOOD	Hierarchical Object Oriented Design
IBM/4LDM	The Four Level Design Method
IEM	Information Engineering Methodology
ISAC	Informations Systems Work and Analysis of Changes
IStar	Integrated Project Support Environment
JSD	Jackson System Development
MASCOT	Modular Approach to Software Construction, Operation and Test
MBOOD	Model-Based Object-Oriented Design
MINI-ASYST	The MINI-ASYST Development Methodology
MULTI/CAM	MULTI/CAM - SDM/STRUCTURED
ObjectOry	

OOA	Object Oriented Analysis
OOA/ST	Object Oriented Analysis
OOD	Object Oriented Design
PAISLey	Process-oriented, Applicative, Interpretable Specification Language
PDL/81	Program Design Language/81
PRIDE	"PRIDE" Family of Products for Information Resource Management (IRM)
PROMOD	the Project Models
PROTOB	
PSL/PSA	Problem Statement Language/Problem Statement Analyzer
RM	Refinement Method
SADT	Structured Analysis and Design Technique
SCR	Software Cost Reduction
SD	Structured Design
SEM	System Engineering Methodology
SSD	Hatley/Pirbhai - Strategies for System Development
STATEMATE	
StP	Software through Pictures
STRADIS	Structured Analysis, Design and Implementation of Information Systems
TAGS	Technology for the Automated Generation of Systems
UCRA	User-Centered Requirements Analysis
WARD/MELLOR	Ward/Mellor Real-Time Method

This page is intentionally blank.

6.1 DEVELOPMENT ACTIVITIES ADDRESSED BY THE MEΓHOD

In question 1 of the survey, developers were asked to describe the level of support which is provided by the method for various activities associated with the development of software systems. Included in the list of activities were requirements definition and clarification, system specification, system design, system implementation and installation, software quality assurance, risk and cost assessment, and other aspects of project management.

The intent of the question was to provide information on the scope of activities addressed by the method. Experience has shown that the level at which activities are addressed by methods can vary widely. Some methods provide detailed instructions for accomplishing a particular activity. Other methods supply a framework to guide the software developer in performing the activity, such as specifying templates for the documents which are the product of the activity. Accordingly, the respondents were asked to select one of the following levels for each of the activities:

S: The method prescribes specific directions and procedures for conducting the activity.

G: The method provides guidelines or a framework for the activity.

R: The method requires the activity but does not provide directions for accomplishing the activity.

N: The method does not address the activity.

A blank entry in Table 1 indicates either that the method does not address the activity, or that the developer provided no response for that activity.

The reader should realize that the phrase "prescribes specific directions", though providing some distinction between methods, can receive various levels of interpretation. While one method developer may consider directions provided as "specific", another may judge such directions to serve only as guidelines. Furthermore, the needs of the user of the method will further influence the judgment on the level of support. Accordingly, the results presented in Table 1 should serve as a guideline, and can not substitute for an in-depth understanding of the method.

The reader is referred to the Description section in the individual write-ups for any further information which may have been provided by the developer. Additionally, bibliographic references have been provided at the end of most write-ups.

TABLE 1. DEVELOPMENT ACTIVITIES ADDRESSED BY THE METHOD

	REQUIREMENTS DEFINTION/ CLARIFICATION	SYSTEM SPECIFICATION	SYSTEM DESIGN	SYSTEM IMPLEMENTATION/ INSTALLATION	SOFTWARE QUALITY ASSURANCE	RISK/COST ASSESSMENT	OTHER PROJECT MANAGEMENT
ABDLSLCM	S	S	S	G	G	G	G
ADM			S	S	S	S	S
AISLE	G	G	S	S	G		G
BOX STRUC	S	S	S	S	S	G	R
BYRON PDL	G	G	G	S	G		
CORE	S	S			G	S	
DARTS	R	S	S	R	R		
DBO	S	S	S		S	S	
DCDS	S	S	S		S	S	R
DSSAD	S	S	S	G	G		
ESA	S	S	S	R	R	R	G
E-DEV	S	S	S	R	R	R	G
GYPSY	G	S	S	S	S		G
HOOD	G	S	S	S	S		
IBM/4LDM		R	S	G	G	G	G
IEM	S	S	S	G	S	G	S
ISAC	S	S	G	R	R	G	G
ISTAR	R	R	R	G	G	G	S
JSD	R	S	S	S	G		
MASCOT	R	S	S	S	S	G	G
MBOOD	S	S	S	G			
MINI-ASYST	G	G	G	G	G	G	S
MULTI/CAM	S	S	S	S	S	S	S

S = SPECIFIC PROCEDURES PRESCRIBED
G = GUIDELINES PROVIDED
R = REQUIRED; NO DIRECTIONS PROVIDED

890124-001M9-4

TABLE 1. DEVELOPMENT ACTIVITIES ADDRESSED BY THE METHOD (CON'T)

	REQUIREMENTS DEFINITION/ CLARIFICATION	SYSTEM SPECIFICATION	SYSTEM DESIGN	SYSTEM IMPLEMENTATION/ INSTALLATION	SOFTWARE QUALITY ASSURANCE	RISK/COST ASSESSMENT	OTHER PROJECT MANAGEMENT
OBJECTORY	S	S	S	S	S	R	R
OOA	S	S	G		S	G	G
OOA/ST	S	S	G				
OOD			S	G			
PAISley	S	S	G				
PDL/81		G	S				
PRIDE	S	S	S	S	S	S	S
PROMOD	S	S	S	S	G	G	G
PROTOB	S	S	S	S	R		
PSL/PSA	G	G	G	G	G	G	G
RM		G	S	S			
SADT	S	S	S	S	S	S	S
SCR	G	G	G	G	G		G
SD		R	S		G	G	
SEM	S	S	S		G	G	G
SSD	S	S	S	G		R	G
STATEMATE	S	S	G	G	S		G
StP	G	G	G				
STRADIS	S	S	S	S	G	G	G
TAGS	S	S	S	G	G	R	R
UCRA	S	S				G	
WARD/MELL	R	S	S				

S = SPECIFIC PROCEDURES PRESCRIBED
G = GUIDELINES PROVIDED
R = REQUIRED; NO DIRECTIONS PROVIDED

890124-002M9-4

This page is intentionally blank.

6.2 APPROACHES USED BY THE METHOD

Many methods conform to a underlying approach or strategy that suggests a general way of beginning or conducting the development effort. In question 2 of the survey, developers were asked to indicate the type of relationship which exists between the method and a particular approach. The responses to the question are given in Table 2.

The approaches listed in the question included the following: the data-flow oriented approach typically associated with structured analysis/design, data structure-oriented approach, object-oriented approach, control-oriented approach, event-oriented approach, entity-relationship modeling, binary relationship modeling, and functional decomposition. When more specific information regarding the underlying approach of the method was provided by the developer, the information is detailed in the Description section of the individual write-up.

Respondents were asked to circle one of the following codes to indicate which code best reflects the relationship between method and approach.

F: The method is founded upon the approach.

C: The method is compatible with the approach.

I: The method is incompatible with the approach.

U: The relationship of the method to the approach is unknown.

One would expect that a response stating that a method is founded on a particular approach could be a basis for characterizing the method. Methods, however, can advocate multiple views of the software system, and thus it is possible that a method could be founded on more than one approach. Furthermore, it is not necessary that a method be based on any particular approach. Methods which provide a framework, rather than specific directions, typically do not focus on some specific approach to software development.

It should be expected that for many of the methods, it is possible to follow a variety of approaches when using the method. Thus, the response "compatible" occurs quite frequently in the table. On the other hand, it should be considered of significant note when the developer indicates that a particular approach is "incompatible" with the method. Finally, for those cases where no entry appears in the table, the developer either provided no response, or stated that the relationship of the method to the approach was unknown.

TABLE 2. APPROACHES USED BY THE METHOD

	DATA FLOW-ORIENTED APPROACH	DATA STRUCTURE-ORIENTED APPROACH	OBJECT-ORIENTED APPROACH	CONTROL-ORIENTED APPROACH	EVENT-ORIENTED APPROACH	ENTITY-RELATIONSHIP MODELING	BINARY-RELATIONSHIP MODELING	FUNCTIONAL DECOMPOSITION
ABDLSLCM	C	C	C	C	C	C	C	C
ADM	F	C	F	C	C			I
AISLE	C	C	C	C	C	C		C
BOX STRUC	C	C	F	C	C	C	C	C
BYRON PDL	C	C	C	C	C			C
CORE	F	F		F	F	C		F
DARTS	F	I	C	F	F	C		C
DBO			C	F				C
DCDS	C	C	C	F	F	F		F
DSSAD	C	F			C	F		I
ESA	F	F	F	F	F	F	C	C
E-DEV	F	F	F	F	F	F	C	C
GYPSY	C	C		C		C	C	F
HOOD	C	C	F	F	F	C	C	C
IBM/4LDM		C	C	C		C		F
IEM	C	C	C	C	C	F	I	F
ISAC	F	F	C	C	C	C	C	F
ISTAR	C	C	C	C	C	F	F	F
JSD	C	C	F		F	C		C
MASCOT	F	F	F	C	C	C		F
MBOOD	F	C	F	C	F	F	C	C
MINI-ASYST	C	C			C	C	C	C
MULTI/CAM	F	C	C	C	C	F		F

F = FOUNDED ON APPROACH
C = COMPATIBLE WITH APPROACH
I = INCOMPATIBLE WITH APPROACH

89C124-003M9-4

TABLE 2. APPROACHES USED BY THE METHOD (CON'T)

	DATA FLOW-ORIENTED APPROACH	DATA STRUCTURE-ORIENTED APPROACH	OBJECT-ORIENTED APPROACH	CONTROL-ORIENTED APPROACH	EVENT-ORIENTED APPROACH	ENTITY-RELATIONSHIP MODELING	BINARY-RELATIONSHIP MODELING	FUNCTIONAL DECOMPOSITION
OBJECTORY	I		F			C	C	
OOA	C	F	F		C	F		I
OOA/ST	C	C	F	C	C	C	C	I
OOD	C	C	F	C	C	C		I
PAISley	C	C	C	F	C			C
PDL/81	C	C	C	F	C	C	C	F
PRIDE	C	F	C	C	C	C	C	C
PROMOD	F	C	F	F	F	C		F
PROTOB	I	I	F	F	C	C	C	I
PSL/PSA	C	C	C	C	C	F	C	C
RM	C	C	C	C	C			F
SADT	F	C	C	C	C	C	C	F
SCR			C					
SD	C	I	I	C		C		F
SEM	F	C	C	F	F	F		F
SSD	F	F	C	F	C	C	C	F
STATEMATE	C	I	C	C	C	C	I	C
StP	F	C	C	F	C	F	C	C
STRADIS	F	F	C	C	C	F	C	F
TAGS	C	C	C	C	F	C	C	F
UCRA	F		C		C	F		F
WARD/MELL	F		C		F	F	C	I

F = FOUNDED ON APPROACH
C = COMPATIBLE WITH APPROACH
I = INCOMPATIBLE WITH APPROACH

890124-004M9-4

This page is intentionally blank.

6.3 THE RELATIONSHIP OF METHODS TO SOFTWARE PROCESS PARADIGMS

In recent years, several alternatives have been proposed to the classic software process paradigm. In question 3 of the survey, the method developers were asked to state how well the method fits into the context of particular process models. The following models were listed in the questionnaire:

The Waterfall model - the classic software process model with feedback loops.

The Spiral model of B. Boehm - alternative evaluation and risk analysis at increasing levels of elaboration.

Rapid prototyping - use of an executable model to clarify requirements.

Incremental or evolutionary development - iteration through successively more elaborate versions of the system.

The Operational model - simulation of the operational spcifications of the system.

The Transformational model - construction of the system through the application of transformation rules from specification to design to code.

The 4GL model - the use of fourth generation languages to produce a prototype of the system, and automatic code generation for the production system.

The respondents were asked to indicate whether the method was well-suited to the process model, was compatible with the process model, or was inappropriate for use with the process model. Additionally, the choice of "no opinion" was provided. When the entry in the table is blank, either no response was given, or the response was "no opinion."

In the opinion of the authors, the most noteworthy response occurs when the developer states that the method is "inappropriate" for use with the process model. In general, unless a method incorporates key aspects of a particular model, such as the use of prototyping, it is likely that use of the method will not be tightly coupled to some particular model. In such a case, there may be little difference between use of the phrase "well-suited" and that of "compatible." In question 4 of the survey, developers were given the opportunity to state that the most effective use of the method depended upon following one particular software process paradigm. If such is the case, specific information is provided in the Description section of the individual write-ups.

Finally, it should be noted that if a method addresses a single aspect of the development process rather than a range of activities, the issue of fitness of the method to a process model may be relatively unimportant.

TABLE 3. RELATIONSHIP OF METHOD TO SOFTWARE PROCESS PARADIGMS

	WATERFALL MODEL	SPIRAL MODEL	RAPID PROTOTYPING	INCREMENTAL MODEL	OPERATIONAL MODEL	TRANSFORMATIONAL MODEL	4GL MODEL
ABDLSLCM	C	C	W	W	C	C	C
ADM	I	I	C	W	C		
AISLE	W	W	I	W	I	W	I
BOX STRUC	C	W	C	W	C	C	C
BYRON PDL							
CORE	W	W	W	C		I	I
DARTS	W	W	W	W	W	I	I
DBO	W	W	C	W		I	W
DCDS	W	W	W	W	W		I
DSSAD	C		I	W		C	C
ESA	W	W	W	W	C	W	W
E-DEV	W	W	W	W	C	W	W
GYPSY	W	W	C				I
HOOD	W	C	C	C	W	I	
IBM/4LDM	W	W	W	W			I
IEM	W	W	W	C			W
ISAC	W	C	C	W	C	W	W
ISTAR	W	W	W	W	C	C	C
JSD	C		W	C	W	W	C
MASCOT	W	W	W	W	W		I
MBOOD	W	C	I	C	C		
MINI-ASYST	C	C	C	C	C		C
MULTI-CAM	W	C	C	W	I	C	W

W = WELL-SUITED
C = COMPATIBLE
I = INAPPROPRIATE

890124-005M9-4

TABLE 3. RELATIONSHIP OF METHOD TO SOFTWARE PROCESS PARADIGMS (CON'T)

	WATERFALL MODEL	SPIRAL MODEL	RAPID PROTOTYPING	INCREMENTAL MODEL	OPERATIONAL MODEL	TRANSFORMATIONAL MODEL	4GL MODEL
OBJECTORY	C	C	W	W	W	W	C
OOA	W		W	W	C		C
OOA/ST	W	W	C	W	W		
OOD	C	W	W	W	W	I	C
PAISley	C	C	W	C	W	W	C
PDL/81	W	W	C	C	C		I
PRIDE	W	W	W	W			W
PROMOD	W	C	C	C	C	W	C
PROTOB	I	C	W	C	W	C	W
PSL/PSA	W	W	C	W	C	C	C
RM	I	C	C	W			
SADT	W	W	W	W	W	W	C
SCR	W	W	W	W	C		
SD	W	W	W	W			C
SEM	W	W	W	W	W		
SSD	W	W	W	W	W	C	C
STATEMATE	W	W	W	W	W		
StP	C		C		I	I	C
STRADIS	W	C	C	C	C		C
TAGS	C	C	W	W	W	C	C
UCRA	W	W	W	W			W
WARD/MELL	W	W	W	W	W	C	C

W = WELL-SUITED
C = COMPATIBLE
I = INAPPROPRIATE

890124-006M9-4

This page is intentionally blank.

6.4 EXTENT OF USAGE OF THE METHOD

Table 4 presents information associated with the extent of usage of the method. In question 5 of the survey, developers were asked in what year was the method first used for the development of a deliverable system. In questions 7 and 8, developers were asked to estimate the number of delivered systems that have been developed using the method, and the number of organizations that have used the method. The responses to these three questions have been summarized in the table, and provide a basis for assessing the extent to which the method has been used in the development of deliverable systems. The various estimate ranges listed as column headings in the table were the ranges provided as responses in the survey questionnaire.

In comparing responses, it is to be expected that those methods which have been available for a longer period of time will have been used more frequently. There are, however, additional factors besides a mere count which should be considered when assessing the extent of use. For example, methods which have been developed internally within a particular organization, and which are primarily intended for use within that organization are likely to show a lower level of usage than those methods which have been marketed commercially. In addition, methods which are targeted toward the development of very large systems are also likely to show a lower level of usage since development of a single such system may entail several years. Thus, when assessing the information in Table 4, it is advisable to consider also the information supplied in Table 6 regarding the project size for which the method is intended, and to consult the individual method write-up to determine if the method evolved within a given organization with the intent of use by that organization.

Finally, for some of the newer methods, it is reasonable to find that there are several organizations using the method on current projects, but that as of the present time, few systems have been completed and delivered using the method.

TABLE 4. EXTENT OF USAGE

	YEAR FIRST USED	ESTIMATE OF NUMBER OF DELIVERED SYSTEMS DEVELOPED USING METHOD					ESTIMATE OF NUMBER OF ORGANIZATIONS HAVING USED THE METHOD				
		LESS THAN 5	5-20	21-100	101-250	MORE THAN 250	LESS THAN 5	5-20	21-50	51-100	MORE THAN 100
ABDLSLCM	1986		●					●			
ADM	1985	●					●				
AISLE	1986			●						●	
BOX STRUC	1987	●						●			
BYRON PDL	1983			●							●
CORE	1980			●					●		
DARTS	1982		●					●			
DBO	1968					●					●
DCDS	1979		●						●		
DSSAD	1981	●					●				
ESA	1980			●						●	
E-DEV	1981		●					●			
GYPSY	1979		●					●			
HOOD	1987		●					●			
IBM/4LDM	1979		●					●			
IEM	1982					●					●
ISAC	1968					●					●
ISTAR	1986		●					●			
JSD	1982			●					●		
MASCOT	1976					●			●		
MBOOD	1985		●						●		
MINI-ASYST	1986			●						●	
MULTI/CAM	1986					●					●

890124-007M9-4

TABLE 4. EXTENT OF USAGE (CON'T)

	YEAR FIRST USED	ESTIMATE OF NUMBER OF DELIVERED SYSTEMS DEVELOPED USING METHOD					ESTIMATE OF NUMBER OF ORGANIZATIONS HAVING USED THE METHOD				
		LESS THAN 5	5-20	21-100	101-250	MORE THAN 250	LESS THAN 5	5-20	21-50	51-100	MORE THAN 100
OBJECTORY	1968		●					●			
OOA	1982			●						●	
OOA/ST	1988	●					●				
OOD	1984			●						●	
PAISley	1985	●					●				
PDL/81	1973					●					●
PRIDE	1971					●					●
PROMOD	1980					●					●
PROTOB	1987		●					●			
PSL/PSA	1973					●					●
RM	1986	●					●				
SADT	1974				●					●	
SCR	1976	●									
SD	1960					●					●
SEM	1981	●					●				
SSD	1982			●						●	
STATEMATE	1983		●							●	
SIP	1984				●						●
STRADIS	1980					●					●
TAGS	1985			●					●		
UCRA	1985				●						●
WARD/MELL	1982					●					●

890124-008M9-4

This page is intentionally blank.

6.5 APPROPRIATE APPLICATION AREAS

In assessing a method, one of the major areas of concern is the appropriateness of a method for a particular application domain. A major motivating factor for the development of several methods was the need to address specific issues associated with a particular application area. Typical of such methods are those for which the primary focus is the development of a database for an enterprise, or those methods which focus on the particular problems which arise in systems required to meet hard real-time constraints.

In question 9 of the survey, developers were asked to indicate the suitability of the method with respect to each of the following application domains:

Embedded systems and/or process control systems.
Time-critical/real-time systems.
Scientific and/or engineering applications.
System programming (operating system software).
Distributed processing and network systems.
Data processing and database applications.
Expert systems and other artificial intelligence applications.
Image processing or pattern recognition.
Large scale simulation and modeling.

The above list is not intended to represent a disjoint set of application domains. Many application problems will involve two or more of the above areas. Furthermore, there are application domains which may not be easily classified under any of the above. Developers were allowed to list other application areas for which the method may be well-suited, and were also asked to list specific types of applications for which delivered systems were developed using the method. This information is available in the Applicability and Usage section of the individual write- ups.

For specifying the suitability, the following choices were provided: 1) the method is well-suited for use in the application area; 2) the method is compatible for use in the application area; 3) the method is inappropriate for use in the application area; and 4) no opinion on the method's suitability for the application area.

In comparing responses for this question, the reader should keep in mind some of the following ideas. First, methods which provide a framework or general guidelines are likely to be suitable for multiple problem domains, but at the same time provide a lower level of specificity. On the other hand, methods which provide specific instructions on how to achieve system specifications or system design tend to be well-suited to certain applications, and less well-suited to others. In particular, those methods developed precisely for the purpose of addressing specific issues associated with a given application domain, such as time-critical issues, may be less suitable for other problem domains.

Ideally, a developer of a method would like to find that the method is suitable for several problem areas. Accordingly, it is significant to note those cases where the developer is willing to state that the method is "inappropriate" for a particular application area.

A blank entry occurs in Table 5 whenever the developer offered no opinion on the suitability, or did not respond for the particular application area.

TABLE 5. APPROPRIATE APPLICATION AREAS

	EMBEDDED SYSTEMS/ PROCESS CONTROL	TIME-CRITICAL/ REAL-TIME	SCIENTIFIC OR ENGINEERING	SYSTEMS PROGRAMMING	DISTRIBUTED PROCESSING OR NETWORKS	DATA PROCESSING OR DATABASE	EXPERT SYSTEMS/AI	IMAGE PROCESSING OR PATTERN RECOGNITION	LARGE-SCALE SIMULATION/ MODELING
ABDLSLCM	W	W	C	W	C	W			W
ADM	W	W	W		C	W	I		W
AISLE	W		W	W	C	C	C	W	W
BOX STRUC	W	W	W	W	W	W	W	W	W
BYRON PDL	W	W	W	W	C	C	C	W	
CORE	W	W		I	W	I	I		W
DARTS	W	W	C	C	W	I	I	W	C
DBO	W	W	W	W	W	W	W	W	W
DCDS	W	W	W	C	W	C		C	W
DSSAD	C	W	C			W			C
ESA	C	C	C	W	W	W	C		
E-DEV	C	C	C	W	W	W	C		
GYPSY	C		W	W	W	W	C	C	
HOOD	W	W	W	C	W	C	C	C	C
IBM/4LDM	W	W	W	I	W	W	I		
IEM	C	C		C	W	W	C	I	
ISAC	W	W	W	C	C	W	C	C	C
ISTAR	W	W	W	W	W	C			
JSD	W	C	I	C	W	W	I	I	W
MASCOT	W	W		C	W	C	I		W
MBOOD	W	W	W	W	W	W			W
MINI-ASYST	I	I	I	C	C	C			C
MULTI/CAM	W	C	I	W	W	W	I	I	W

W = WELL-SUITED
C = COMPATIBLE
I = INAPPROPRIATE

890124-009M9-4

TABLE 5. APPROPRIATE APPLICATION AREAS (CON'T)

	EMBEDDED SYSTEMS/ PROCESS CONTROL	TIME-CRITICAL/ REAL-TIME	SCIENTIFIC OR ENGINEERING	SYSTEMS PROGRAMMING	DISTRIBUTED PROCESSING OR NETWORKS	DATA PROCESSING OR DATABASE	EXPERT SYSTEMS/AI	IMAGE PROCESSING OR PATTERN RECOGNITION	LARGE-SCALE SIMULATION/ MODELING
OBJECTORY	W	W	W	W	W	W			W
OOA	W	W	W	W	W	W	C	I	W
OOA/ST	W	W	W	C	W	W	C	I	W
OOD	W	W	C	W	W	W	C	C	W
PAISley	W	W	I	W	C	I	I	I	C
PDL/81	W	W	C	W	W			C	
PRIDE	W	W	W	C	W	W	W	C	C
PROMOD	W	W	W	W	W	C	C	W	W
PROTOB	W	W	I	C	W	I	C	I	W
PSL/PSA	W	W	W	W	C	W	C	C	W
RM	W	C	W	W	C	C	I	C	W
SADT	W	C	W	W	W	C	W	C	W
SCR	W	W	C	W	W				
SD	C	C	C	W	C	C	C		C
SEM	W	W	C	C	W	C	W	C	W
SSD	W	W	W	W	W	C	C	C	W
STATEMATE	W	W	W	W	W	C			W
StP	W	W	W	W	W	W	I		I
STRADIS	C	C	C	C	W	W	C	C	C
TAGS	W	W	W	C	W	C	C	C	W
UCRA	I		C		W	W	W		C
WARD/MELL	W	W	C	W	W		C	C	C

W = WELL-SUITED
C = COMPATIBLE
I = INAPPROPRIATE

890124-010M9-4

This page is intentionally blank.

6.6 RELATION OF THE METHOD TO PROJECT SIZE

In question 11 of the survey, developers were asked to describe the relationship of the method to project size by specifying for what size projects the method is intended, and for what size projects the method has been used. The responses have been summarized in Table 6.

The size of projects was characterized in the choice of answers by:

S: Small projects, e.g., a controller for a bank of elevators.

M: Medium projects, e.g., a communication network manager.

L: Large projects, e.g., a software system for a space station.

For some methods, the issue of project size may not represent a major factor. The use of other methods, however, may be optimal for a certain size of project. In particular, there are those methods which have been developed specifically to deal with the special issues that arise in the development of very large systems. In such development efforts, communication, project management, and changes in the personnel and the environment associated with the development become much more critical than for smaller size projects. Thus, the size of project for which the method is intended is an issue of significance, and may be an important factor to consider when selecting a new method. The intended size may also be a consideration when determining the suitability of the method for a particular application. Additionally, information concerning the size of projects for which the method has been used supplements the information which is supplied in Table 4 relative to the usage of the method.

As a final comment, the reader should note that the terms small, medium, and large are relative terms, and as such are subject to wide interpretation. More specific size data, such as lines of code or number of man-months associated with a project, would be more informative, but the authors judged that such information might not be readily available to the persons answering the questionnaire.

TABLE 6. RELATION OF METHOD TO PROJECT SIZE

	PROJECT SIZE SUITABLE FOR USE WITH METHOD			PROJECT SIZE FOR WHICH METHOD HAS BEEN USED		
	SMALL	MEDIUM	LARGE	SMALL	MEDIUM	LARGE
ABDLSLCM	●	●	●	●	●	
ADM	●	●	●			●
AISLE	●	●	●	●	●	●
BOX STRUC	●	●	●	●	●	●
BYRON PDL		●	●		●	●
CORE	●	●	●	●	●	●
DARTS	●	●	●	●	●	
DBO	●	●	●	●	●	●
DCDS			●			●
DSSAD	●	●	●	●	●	
ESA	●	●	●	●	●	●
E-DEV	●	●	●	●	●	●
GYPSY		●		●	●	
HOOD		●	●	●	●	●
IBM/4LDM			●		●	●
IEM		●	●	●	●	●
ISAC		●	●	●	●	●
ISTAR		●	●		●	
JSD	●	●	●	●	●	
MASCOT		●	●	●	●	●
MBOOD	●	●	●	●	●	
MINI-ASYST	●	●	●	●	●	●
MULTI/CAM	●	●	●	●	●	●

890124-011M9-4

TABLE 6. RELATION OF METHOD TO PROJECT SIZE (CON'T)

	PROJECT SIZE SUITABLE FOR USE WITH METHOD			PROJECT SIZE FOR WHICH METHOD HAS BEEN USED		
	SMALL	MEDIUM	LARGE	SMALL	MEDIUM	LARGE
OBJECTORY		●			●	●
OOA	●	●	●	●	●	●
OOA/ST	●	●	●	●	●	
OOD	●	●	●	●	●	●
PAISley				●	●	
PDL/81	●	●		●	●	●
PRIDE	●	●	●	●	●	●
PROMOD	●	●	●	●	●	●
PROTOB		●			●	
PSL/PSA	●	●	●	●	●	●
RM			●		●	
SADT		●	●	●	●	●
SCR	●	●	●	●	●	
SD	●	●	●	●	●	●
SEM		●	●		●	●
SSD	●	●	●	●	●	●
STATEMATE		●	●	●	●	●
StP	●	●	●	●	●	●
STRADIS	●	●	●	●	●	●
TAGS		●	●	●	●	
UCRA	●	●	●	●	●	●
WARD/MELL		●	●		●	●

890124-012M9-4

This page is intentionally blank.

6.7 RELATIONSHIP OF METHODS TO PROGRAMMING PRACTICES/CONCEPTS

Question 13 involved the relationship of the method to various concepts or practices associated with modern program development. Developers were asked to select one of the following for each item in the list of concepts/practices.

E: The concept/practice is essential to the method.

C: The concept/practice is compatible with the method.

I: The concept/practice is inconsistent with the method.

U: The relationship of the concept/practice to the method is unknown.

The list of concepts/practices included the following: stepwise refinement, information hiding, process abstraction, abstract data-types, structured programming, genericity, inheritance, use of pre- and post-condition assertions, and module coupling/cohesion. Though the questionnaire used only brief phrases for the concepts and practices, the authors believe that these concepts are well-known in the software engineering community, and that the intent of the reference was clear. For example, module coupling/cohesion refers to the practice of evaluating module structures of a system to strive for a low level of coupling and a high level of cohesion.

Examining the relationship of the method to such concepts and practices should provide a basis of judging the fitness of a method to the development procedures of an organization, as well as judging how well the method will fit the source language in which a system is to be developed. For example, for a system which is to be developed in Ada, one would choose a method which employs, or is compatible with, information hiding, abstract data-types, structured programming, and genericity.

The entry in a table has been left blank whenever the developer selected the response "unknown", or made no choice for the given concept or practice.

TABLE 7. RELATIONSHIP OF METHOD TO PROGRAMMING PRACTICES/CONCEPTS

	STEPWISE REFINEMENT	INFORMATION HIDING	PROCESS ABSTRACTION	ABSTRACT DATA-TYPES	STRUCTURED PROGRAMMING	GENERICITY	INHERITANCE	USE OF ASSERTIONS	MODULE COUPLING/COHESION
ABDLSLCM	C	C	C	C	C	C			C
ADM	E	E	E	E	E	E	C	C	E
AISLE	E	C	C	C	E	C	C	I	C
BOX STRUC	E	E	E	E	E	C	C	C	C
BYRON PDL	C	C		C	C	C			
CORE	E	I		I	C		I	C	C
DARTS	C	E	C	E	E			C	E
DBO	E	C				C	C	C	
DCDS	E	E	E	E	E	C		C	E
DSSAD				C	E				C
ESA	C	C	E	E	C		C	C	E
E-DEV	C	E	E	E	C		E	C	E
GYPSY	C	C	E	C	C	I		E	
HOOD	E	E	E	E	E	C	C	C	C
IBM/4LDM	E	E	E	E	E			C	C
IEM	E	E	E	E	C	E	E	E	E
ISAC	E	C	E	C	E	C	C	C	E
ISTAR	C	C	C	C	C	C	C	C	C
JSD	I	C	E	C	E		C	I	C
MASCOT	C	E	C	E	C	E	C	C	E
MBOOD	C	E		E	C	E	C		C
MINI-ASYST	C		C	C	C				C
MULTI/CAM	C	C	E	C	E	C	C	I	E

E = ESSENTIAL
C = COMPATIBLE
I = INCONSISTENT

890124-013M9-4

TABLE 7. RELATIONSHIP OF METHOD TO PROGRAMMING PRACTICES/CONCEPTS (CON'T)

	STEPWISE REFINEMENT	INFORMATION HIDING	PROCESS ABSTRACTION	ABSTRACT DATA-TYPES	STRUCTURED PROGRAMMING	GENERICITY	INHERITANCE	USE OF ASSERTIONS	MODULE COUPLING/COHESION
OBJECTORY	C	E	E	E	C	C	E	C	E
OOA	I	E	C	E			E	C	E
OOA/ST	C	E	E	E		C	C	C	
OOD	C	E	C	E	C	E	E	C	E
PAISley	E	C	E	C		C	I		
PDL/81	E	C	E		E	C			C
PRIDE	E		E	E	C				C
PROMOD	E	E	E	E	E	C	C	C	E
PROTOB	C	E	E	E	E	E	E	I	I
PSL/PSA	C	C	C	C	C	C	C	C	C
RM	E	C	C	C	C	C	C		E
SADT	E	C	E	E	C			C	C
SCR	E	E	E	E	E		C	C	C
SD	E	E		C	C		C	C	E
SEM	E	E	E	C	C	C		C	C
SSD	E	E	E	C	C	C	C	C	C
STATEMATE	C	C	C	C	C	C		C	C
SIP	C	C	E	C		C	C	C	E
STRADIS	E	C	E	C	E	C	C	C	E
TAGS	E	C	E	C	I	C	C	C	E
UCRA	E	C	C	C	C	U	C	C	C
WARD/MELL		C		C	C	C	C	C	C

E = ESSENTIAL
C = COMPATIBLE
I = INCONSISTENT

890124-014M9-4

This page is intentionally blank.

6.8 ABILITY TO ADDRESS REQUIREMENTS OF THE TARGET SYSTEM

In determining the suitability of a method for a particular application problem, one significant feature of a method is the facilities provided which allow the system developers to address specific constraints associated with the target system. In question 23, the method developers were asked whether the method prescribes steps for handling the following types of requirements or constraints of the target system: timing constraints, spatial constraints, special features of the target hardware architecture, special features of the target operating system, concurrency issues, fault-tolerance issues, and security of access. A summary of the responses is provided in Table 8.

The authors' experience with the survey questionnaire for the first edition of the catalog has shown that the phrase "prescribes steps" can receive a wide variety of interpretations. At one end of the spectrum there are those methods which provide specific steps intended to address particular constraints, such as precise ways to address timing limitations. In contrast, there are methods which prescribe, or may only suggest, that specific constraints must be addressed, but which do not provide any directions relative to how this is to be accomplished. In an effort to determine the level at which a method addresses specific requirements, method developers were asked to give a brief description of how a method addresses the particular requirement or constraint. When such information was provided, an asterisk has been placed next to the Y code, and the reader is referred to the Target Constraints section of the individual write-up for the method.

A blank entry has been recorded in the table when no response was provided by the developer.

TABLE 8. ABILITY TO ADDRESS REQUIREMENTS OF THE TARGET SYSTEM

	TIMING CONSTRAINTS	SPATIAL CONSTRAINTS	HARDWARE ARCHITECTURE FEATURES	OPERATING SYSTEM FEATURES	CONCURRENCY ISSUES	FAULT-TOLERANCE ISSUES	SECURITY OF ACCESS
ABDLSLCM	Y*	Y*	Y*	Y*	Y*	Y*	Y*
ADM	Y*	N	N	N	Y*	Y*	N
AISLE	Y*	Y*	N	N	Y*	N	N
BOX STRUC	Y*	Y*	Y*	Y*	Y*	Y*	Y*
BYRON PDL						N	
CORE	N	N	N	N	Y*	Y*	N
DARTS	Y*	N	Y*	Y*	Y*	N	N
DBO	Y	Y	Y	Y			
DCDS	Y*	Y*	Y*	Y*	Y*	Y*	N
DSSAD	N	N	N	N	N	N	N
ESA	N	N	Y	Y	N	N	N
E-DEV	Y	N	Y	Y	N	Y	Y
GYPSY	N	Y*	Y*	N	Y*	N	Y*
HOOD	Y*	Y*	Y*	Y*	Y*	Y*	N
IBM/4LDM	Y*	Y*	Y*	Y*	Y*	Y*	N
IEM	N	N	Y*	N	Y*	N	Y*
ISAC	Y	Y	Y	N	N	N	Y
ISTAR	N	N	N	N	N	N	N
JSD	N	N	Y*	N	Y*	N	N
MASCOT	Y*	Y*	Y*	Y*	Y*	Y*	N
MBOOD	N	N	Y*	Y*	Y*	Y*	N
MINI-ASYST	Y*	Y*	Y*	Y*	Y*		Y*
MULTI/CAM	Y	Y	Y	Y	Y	N	Y

Y = YES; REFER TO METHOD WRITEUP FOR MORE INFORMATION*
Y = YES
N = NO

890124-015M9-4

TABLE 8. ABILITY TO ADDRESS REQUIREMENTS OF THE TARGET SYSTEM (CON'T)

	TIMING CONSTRAINTS	SPATIAL CONSTRAINTS	HARDWARE ARCHITECTURE FEATURES	OPERATING SYSTEM FEATURES	CONCURRENCY ISSUES	FAULT-TOLERANCE ISSUES	SECURITY OF ACCESS
OBJECTORY							
OOA	Y	N	N	N	Y*	Y*	N
OOA/ST	N	N	N	N	Y*	Y*	N
OOD	Y*	Y*	Y*	Y*	Y*	Y*	Y*
PAISley	Y	Y	N	N	Y	N	N
PDL/81	N	N	N	N	N	N	N
PRIDE	Y*		Y	Y	Y*	N	Y
PROMOD	Y*	Y*	Y*	Y*	Y*	Y*	Y*
PROTOB	Y*	N	N	N	Y*	N	N
PSL/PSA	Y	Y	Y	Y	Y	Y	Y
RM	N	N	N	N	Y	N	N
SADT							
SCR	Y*	N	Y*	Y*	Y*	Y*	Y*
SD	N	N	N	N	N	N	N
SEM	Y*	N	Y*	N	Y*	Y*	N
SSD	Y	Y	Y	Y	Y	Y	Y
STATEMATE	Y*		Y*	Y*	Y*	Y*	Y*
StP	Y*	Y*	Y*	Y*	N	Y	N
STRADIS	Y*	N	Y*	N	N	N	N
TAGS	Y*	N	N	N	Y*	Y*	Y
UCRA	Y*	N	Y*	Y*	N	N	Y
WARD/MELL	Y*	N	N	Y*	Y*	Y*	Y*

Y = YES; REFER TO METHOD WRITEUP FOR MORE INFORMATION*
Y = YES
N = NO

890124-016M9-4

This page is intentionally blank.

6.9 TEXTUAL MODES OF COMMUNICATION

In comparing methods, one is often faced with the difficulty that other components of the software process may also affect the aspect being compared. Such is the case, for example, if one wants to contrast methods based upon the reliability of the software produced. Obviously there are other components of the software process, especially the development team itself, which will have an effect on reliability.

One feature which is clearly associated with a method itself and which is not dependent upon other components of the software process is the modes used by the method to represent the evolving software system. This aspect may also represent a significant factor when a given organization is selecting a new method, since the organization may have extensive experience in the use of a particular representation.

In question 15 of the survey, method developers were asked to indicate the extent to which a method uses various modes of representation. Was use of the mode required by the method, was it strongly encouraged, or was its use compatible with the method? Developers were also allowed to specify that use of the mode was inconsistent with the method, or that the relationship of the mode to the method was unknown.

In Table 9, the responses of the developers are summarized for those modes which are largely textual. These include the use of specified documentation templates, narrative overviews of modules, structured English, program design languages, mathematical notation, Warnier/Orr diagrams, and decision tables. The developer was also allowed to list other modes. When an entry occurs in the Other column, readers should refer to the Modes of Expression section in the individual write-up for additional information. Iconographical modes of communication are listed in Table 10.

In addition to specifying the extent to which the method employs the mode, developers were also asked to indicate if automated support for the particular mode is supplied with the method. When the developer stated that such support is provided, an asterisk has been placed in the table behind the appropriate code.

In the authors' opinion, it is significant to note when a method either requires or strongly encourages the use of a particular mode. It is also significant to note when the use of a mode is inconsistent with the method. A blank entry is recorded in the table if the developer indicated that the relationship of the mode to the method was unknown, or if no response was provided for the given mode.

TABLE 9. TEXTUAL MODES OF COMMUNICATION

	SPECIFIED DOCUMENTATION TEMPLATE/S	NARRATIVE OVERVIEWS OF MODULES	STRUCTURED ENGLISH	PROGRAM DESIGN LANGUAGE	FORMAL SPECIFICATION LANGUAGES	MATHEMATICAL NOTATION	WARNIER/ORR DIAGRAMS	DECISION TABLES	OTHER
ABDLSLCM	R*	R*	E*	R*	R*	E*		C	
ADM	C	C	I	R	E	C			
AISLE	R*	R*	R*	R*	I	C	I	I	
BOX STRUC	C	C	R*	R*	R*	R*	I	C	
BYRON PDL	C	R	C	R*	C			I	
CORE	E	E	E	C					
DARTS	E	E	R	E	C	C	I	E	
DBO								C	R
DCDS	R*	R*	I	E*	R*	I	I	E*	
DSSAD	R	E	E	E		I	I	I	
ESA			R						
E-DEV			R						
GYPSY	C	C	C	R*	R*	C			
HOOD	E*	R*	E	R*	C	C	C	C	
IBM/4LDM	R	C	C	R	E	E			
IEM		C	E	I	E	C	C	I	R
ISAC	R*	E	E	E	E	C	C	R	
ISTAR	E	E	C	C	E	E	C	C	
JSD	E*	E*	I	I	R*		I	C	
MASCOT	E	R	E	E	C	C	I	C	R*
MBOOD		C	C	C	C	C	I	E	
MINI-ASYST	E	E	C	C	C		C	C	
MULTI/CAM	R	R	R	E	E	E	C	E	

R = REQUIRED
E = STRONGLY ENCOURAGED
C = COMPATIBLE
I = INCONSISTENT
** = AUTOMATED SUPPORT PROVIDED*

890124-017M9-4

TABLE 9. TEXTUAL MODES OF COMMUNICATION (CON'T)

	SPECIFIED DOCUMENTATION TEMPLATE/S	NARRATIVE OVERVIEWS OF MODULES	STRUCTURED ENGLISH	PROGRAM DESIGN LANGUAGE	FORMAL SPECIFICATION LANGUAGES	MATHEMATICAL NOTATION	WARNIER/ORR DIAGRAMS	DECISION TABLES	OTHER
OBJECTORY	R*	R*	R*	R*	R*		I	I	
OOA	E			C		E			
OOA/ST	C	C	C	C	C	C	I	C	
OOD	R	R	C	R	E	E	I	C	
PAISley	C	C	C	C	R*	R*	I	C	
PDL/81	C*	E*	R*	R*	I	C	I	C	
PRIDE	R*	E*	C	C	C	C	C	C	
PROMOD	E*	E*	E*	E*	E*	C	I	E	
PROTOB	E*		I	I	I	I	I	I	R
PSL/PSA	C*	C*	C*	C*	R*	C	C	C*	
RM	C		C	C					
SADT	C	E	C	C	C	C	C	C	
SCR	E	I			R	R			R
SD	C	C	C	I	C	C	C	C	
SEM	R*	C*	C*	C*	C*	R*		R*	
SSD	R	R	E	C	C	E	I	R	
STATEMATE	E*	C*	C		E*	C*			
StP	E*	E*	E*	E	C	C	I	E*	
STRADIS	R	R*	R	E	C	C	C	E	
TAGS	E*	C	C	C	R*	R*	C	C	
UCRA	E	R	E					E	
WARD/MELL	C*	C*	C*	C*		C	I	C*	

R = REQUIRED
E = STRONGLY ENCOURAGED
C = COMPATIBLE
I = INCONSISTENT
** = AUTOMATED SUPPORT PROVIDED*

890124-018M9-4

This page is intentionally blank.

6.10 ICONOGRAPHICAL MODES OF COMMUNICATION

Table 10 is a continuation of responses associated with modes of communication used by the method. Textual modes were presented in Table 9. In this table are listed those modes which are largely iconographical. These include: finite-state diagrams, Petri nets, data-flow diagrams, control-flow diagrams, entity-relationship diagrams, flowcharts, HIPO charts, hierarchy charts, Leighton diagrams, Nassi-Shneiderman charts, Buhr diagrams, and Booch diagrams. In classifying the representations, the authors recognize that all of these representations also include textual material.

It was not intended that the above list include all forms of iconographical modes, and the developers were given the opportunity to list other modes. When an entry occurs under the column headed "Other", the reader is referred to the individual write-up for additional information.

As with the textual modes, developers were asked to specify which of the following best describes the relationship of the mode to the method. Is use of the mode required by the method? Is its use strongly encouraged, compatible, or inconsistent with the method? Or is the relationship of the mode to the method unknown?

The responses of the developers are summarized in Table 10. If the developer did not respond, or indicated that the relationship is unknown, a blank entry occurs in the table. As in Table 9, if automated support for the mode is provided with the method, an asterisk is recorded with the code in the table.

TABLE 10. ICONOGRAPHICAL MODES OF COMMUNICATION

	FINITE-STATE DIAGRAMS	PETRI NETS	DATA-FLOW DIAGRAMS	CONTROL-FLOW DIAGRAMS	ENTITY-RELATIONSHIP DIAGRAMS	FLOWCHARTS	HIPO CHARTS	HIERARCHY CHARTS	LEIGHTON DIAGRAMS	NASSI-SHNEIDERMAN CHARTS	BUHR DIAGRAMS	BOOCH DIAGRAMS	OTHER
ABDLSLCM	C	C	R*	E*	E*	C	C	R*		C		C	E*
ADM	E	R	R	R	C			R			R	C	R
AISLE	I	I	R	R	C	I	I	I	I	I	E*	E*	
BOX STRUC	E*	C	I	I	E*	C	I	R*		C		C	
BYRON PDL	I	I	I	I	I	I	I	I	I	I	I	I	
CORE			R	R		C							
DARTS	R	C	R	H	E	I	I	R		C	C	C	
DBO													E
DCDS	C*	C	E*	R*	E*	R*	C*	C*		I	C	C	
DSSAD		I	R		R	I	I	R	I				
ESA	E		R	E	R	C	C	E					C
E-DEV			R	E	R	C	C	R		E			R
GYPSY	C	C	C	C		C	C	C	C	C			
HOOD	E	E	R	R	C	C	C	C	C	C	C	C	E
IBM/4LDM	R	C	C	C	C	I	I	C				C	
IEM	E	C	R	C	R	I	C	R		I			ER
ISAC	C	C	R	E	E	C	C	R		C			
ISTAR	C	C	C	C	C	C	C	C	C	C	C	R*	
JSD					C								R*
MASCOT	E		R	C		C	I	R			I	I	
MBOOD	E		E	E	E	I		C			C	C	
MINI-ASYST	I		E	C	E	E	C	C					
MULTI/CAM	E	E	R	E	E	E	C	E	C	C	C	C	

R = REQUIRED
E = STRONGLY ENCOURAGED
C = COMPATIBLE
I = INCONSISTENT
** = AUTOMATED SUPPORT PROVIDED*

890124-019M9-5

TABLE 10. ICONOGRAPHICAL MODES OF COMMUNICATION (CON'T)

	FINITE-STATE DIAGRAMS	PETRI NETS	DATA-FLOW DIAGRAMS	CONTROL-FLOW DIAGRAMS	ENTITY-RELATIONSHIP DIAGRAMS	FLOWCHARTS	HIPO CHARTS	HIERARCHY CHARTS	LEIGHTON DIAGRAMS	NASSI-SHNEIDERMAN CHARTS	BUHR DIAGRAMS	BOOCH DIAGRAMS	OTHER
OBJECTORY	R*	C	I		C		I	R		I	I	C	R*
OOA	R	C	R	C	R			C					
OOA/ST	E		E	C	E	I	I	C		I	C	C	E
OOD	R	R	E	E	E	I	I				R	R	
PAISley	C	C	C	C	C	C							
PDL/81	C	C	C	C	C	I	I	I		I		C	
PRIDE	C	C	C	R*	I	R*	C	R*	C	C	C	C	R
PROMOD	E*	I	E*	E*	I	I	I	E*		I	C	C	
PROTOB	I	R*	I	I	I	I	I	I	I	I	I	I	
PSL/PSA	C*	C	C*	C*	C	C*	C*	C*	C	C	C	C	
RM			C	C		C		E			C	C	
SADT	E*	C	R*	R*	C	C	C	R					
SCR	E	C	C	C	C	I	I	I					E
SD	C	I	C	R	C	C	C	R		C		I	
SEM		E*	R*	R*	R*	I	I	C*		I	C		
SSD	R	C	R	R	C	C		C		C	C	C	
STATEMATE	R*		C*	C*	C*	C	C	C*					*
StP	E*	I	E*	E*	E*	I	I			I	I	I	
STRADIS	E	C	R	C	R	E	C	E		C			
TAGS	C	E	R	R	C	E	E	E	C	C	C	C	
UCRA			R		R		I	R					
WARD/MELL	R*		R*	R*	R*	C	I	R*		C	C*		

890124-020M9-5

R = REQUIRED
E = STRONGLY ENCOURAGED
C = COMPATIBLE
I = INCONSISTENT
** = AUTOMATED SUPPORT PROVIDED*

This page is intentionally blank.

6.11 TECHNIQUES FOR REQUIREMENTS CLARIFICATION

Various techniques are employed in modern software development to assist the development team in clarifying the system requirements or in acquiring early information relative to expected system behavior. Included in these techniques are rapid prototyping, dynamic animation, simulation, incremental and evolutionary development, and the use of executable specifications. In question 14a of the survey, developers were asked to indicate the extent to which a method utilizes the above techniques. The following choices were provided for the responses:

R: The technique is required/essential to the method.

E: The technique is strongly encouraged by the method.

N: The technique is not addressed by the method.

A summary of the responses is provided in Table 11. If the developer indicated that the technique is not addressed by the method, or if no response was given, a blank entry occurs in the table.

The developer was also given the opportunity to list other techniques which are used by the method to clarify system requirements. Where such techniques were listed, the information has been provided in the corresponding section of the individual write-up.

TABLE 11. TECHNIQUES FOR REQUIREMENTS CLARIFICATION

	RAPID PROTOTYPING	DYNAMIC ANIMATION	SIMULATION	INCREMENTAL/ EVOLUTIONARY DEVELOPMENT	EXECUTABLE SPECIFICATIONS
ABDLSLCM	E	E	E	R	E
ADM	R			R	R
AISLE	E			R	
BOX STRUC	E	E	E	E	E
BYRON PDL					
CORE		E	E		
DARTS	E	E	E	E	
DBO	E			R	
DCDS			R	E	R
DSSAD				R	R
ESA	R			E	
E-DEV	R			E	E
GYPSY	E			E	
HOOD		E	E	R	E
IBM/4LDM	E			E	E
IEM	E	E		E	E
ISAC	E	E	E	R	R
ISTAR	E			R	
JSD	E	E	E		R
MASCOT		E		R	
MBOOD					
MINI-ASYST	E	E	E	E	
MULTI/CAM	E	E	E	E	E

R = REQUIRED/ESSENTIAL
E = STRONGLY ENCOURAGED

890124-021M9-5

TABLE 11. TECHNIQUES FOR REQUIREMENTS CLARIFICATION (CON'T)

	RAPID PROTOTYPING	DYNAMIC ANIMATION	SIMULATION	INCREMENTAL/ EVOLUTIONARY DEVELOPMENT	EXECUTABLE SPECIFICATIONS
OBJECTORY	E	E	E	E	E
OOA				R	
OOA/ST				E	
OOD	E	E	E	R	E
PAISley	R	E	R	E	R
PDL/81				E	
PRIDE	E		E	R	E
PROMOD	E			E	
PROTOB	R	R	R	E	R
PSL/PSA	E	E	E	E	E
RM	R			R	E
SADT	E	E	E	E	E
SCR	E			E	
SD					
SEM	R	E	R	E	E
SSD	E	E	E	R	E
STATEMATE	E	E	E	E	E
StP	E				
STRADIS	E	E	E		
TAGS	R	E	R	E	R
UCRA				E	
WARD/MELL		E	E	E	R

R = REQUIRED/ESSENTIAL
E = STRONGLY ENCOURAGED

890124-022M9-5

This page is intentionally blank.

6.12 ANALYSIS AND REVIEW TECHNIQUES USED BY THE METHOD

Major components in the development of software are the techniques used to analyze the system and to review the evolving software. Included in these techniques are data-structure analysis, data-flow analysis, control-flow analysis, the use of decision tables, formal proof techniques, design reviews, code walk-throughs, Facilitated Application Specification Techniques, and Change Control Board review. In question 14b of the survey, developers were asked to indicate the extent to which a method utilizes each of the above techniques. The following choices were provided for the responses:

R: The technique is required/essential to the method.

E: The technique is strongly encouraged by the method.

N: The technique is not addressed by the method.

A summary of the responses is provided in Table 12. If the developer indicated that the technique is not addressed by the method, or if no response was given, a blank entry occurs in the table.

The developer was also given the opportunity to list other techniques which are used by the method to perform analysis and review. Where such techniques were listed, the information has been provided in the corresponding section of the individual write-up.

TABLE 12. ANALYSIS AND REVIEW TECHNIQUES

	DATA-STRUCTURE ANALYSIS	DATA-FLOW ANALYSIS	CONTROL-FLOW ANALYSIS	DECISION TABLES	FORMAL PROOF TECHNIQUES	DESIGN REVIEWS	CODE WALK-THROUGHS	FACIL. APP. SPEC. TECH.	CHANGE CONTROL BOARD REVIEW
ABDLSLCM	E	E	E	E	E	R	R	E	E
ADM		R	R			R	R		R
AISLE	R	E	E			R	R		E
BOX STRUC	E			E	R	R	E	E	E
BYRON PDL									
CORE	R	R	R						
DARTS	E	R	R	E		R	E		E
DBO									E
DCDS	R	R	R	E		R	R		E
DSSAD	R	E	E			R	R		
ESA	R	R	R	R				R	E
E-DEV	R	R	R	R	E	R	E	R	E
GYPSY	E	E	E		R				
HOOD	E	E	E			R	R	E	R
IBM/4LDM	E	E	E		E	R	R	E	E
IEM	R	R	R		E	R	E	R	E
ISAC	R	R	R	R	E	E	E	R	E
ISTAR					E	E	E		R
JSD						E			
MASCOT		R	E	E		R	E		E
MBOOD	E	E	E	E		E	E		
MINI-ASYST	E	E	E	E	E	E	E	E	E
MULTI/CAM	E	R	E	E	E	E	E	E	E

R = REQUIRED/ESSENTIAL
E = STRONGLY ENCOURAGED

890124-023M9-5

TABLE 12. ANALYSIS AND REVIEW TECHNIQUES (CON'T)

	DATA-STRUCTURE ANALYSIS	DATA-FLOW ANALYSIS	CONTROL-FLOW ANALYSIS	DECISION TABLES	FORMAL PROOF TECHNIQUES	DESIGN REVIEWS	CODE WALK-THROUGHS	FACIL. APP. SPEC. TECH.	CHANGE CONTROL BOARD REVIEW
OBJECTORY						E	E		R
OOA	R	R	R			E	E		E
OOA/ST	R	R	R	E		E			
OOD	E	E	E	E	E	E	E		E
PAISley	E	E	E						
PDL/81			R			E	E		E
PRIDE	R	R	R	E	R	R		R	E
PROMOD	R	R	R	E		E	E	E	E
PROTOB		E	E			E	E		
PSL/PSA	E	E	E	E	E	E	E	E	E
RM		E	E			E	R		
SADT		R	R	E	E	R	E	E	E
SCR					E	R			
SD	R	E	R						
SEM	E	R	R	R		E			
SSD	R	R	R	R	E	E	E		E
STATEMATE		E	E		E	E			
StP	E	E	E	E		E	E		
STRADIS	R	R	E	E		R	R		R
TAGS	E	R	R	E		E	E	R	R
UCRA	R	E		E		E		E	E
WARD/MELL	E	E	E	E		E			

R = REQUIRED/ESSENTIAL
E = STRONGLY ENCOURAGED

890124-024M9-5

This page is intentionally blank.

6.13 PROJECT MANAGEMENT SUPPORT PROVIDED BY METHOD

In question 24 of the survey, developers were asked to indicate the level of support provided by the method for various project management activities. A summary of the responses is provided in Tables 13A and 13B. As can be seen from the large number of blank entries in the tables, many of the methods described in this catalog do not address any aspect of project management, or address only a few of the activities associated with project management.

Table 13A summarizes the responses for those activities associated with risk assessment and cost estimation. These activities include analyzing risk, assessing complexity, estimating initial cost, projecting cost of completion, providing incremental data about expenditures, and tracking project progress. Table 13B summarizes the responses for other activities associated with project management. These include project planning, scheduling and/or manpower loading, generation of Pert or Gantt charts, allocation of personnel to tasks, allocation of development resources, configuration management, and reliability estimation. The developer was also given the opportunity to list other activities which the method may support. When an entry occurs under the column headed "Other", the reader is referred to the Project Management section of the individual write-up for additional information.

For each of the activities, the developer was asked to specify the level of support provided by the method by selecting one of the following:

S: The method prescribes specific directions and procedures for conducting the activity.

G: The method provides guidelines or a framework for the activity.

R: The method requires the activity but does not provide directions for accomplishing the activity.

N: The method does not address the activity.

Additionally, the developer was asked to indicate when there are automated features of the method which support the particular activity. The availability of such automated support is indicate in the tables by use of an asterisk. Finally, a blank entry appears in the table if the developer specified that the method does not address the activity, or if the developer did not respond for the particular activity.

TABLE 13A. PROJECT MANAGEMENT SUPPORT

	RISK ANALYSIS	COMPLEXITY ASSESSMENT	INITIAL COST ESTIMATES	COST-TO-COMPLETE PROJECTION	INCREMENTAL DATA ABOUT EXPENDITURES	PROJECT TRACKING	OTHER
ABDLSLCM	G*	G*	G	G	G	G*	G
ADM	G	S				S	
AISLE	S*	S*				S*	
BOX STRUC	G	G	G	G	G	G	
BYRON PDL							
CORE	S		R	R			
DARTS							
DBO	S*		S*	S*	S*	S*	S
DCDS	S*					S*	
DSSAD		G					
ESA	S	S	S	S	S	S	
E-DEV	S	S	S	S	S	S	
GYPSY							
HOOD							
IBM/4LDM	S	S	S	G	G	G	
IEM		G	R	R		G	
ISAC	G	G	S	S	S	G	
ISTAR	S*	S*	S*	S*	S*	S*	S*
JSD		G					
MASCOT	G	G	S	G	G	S*	
MBOOD		G					
MINI-ASYST	G	G	G	G	G	G	
MULTI/CAM	S*	S*	S*	S*	S*	S*	

S = SPECIFIC DIRECTIONS/PROCEDURES PRESCRIBED
G = GUIDELINES/FRAMEWORK PROVIDED
R = REQUIRED; NO DIRECTIONS PROVIDED
** = AUTOMATED SUPPORT PROVIDED*

890124-025M9-5

TABLE 13A. PROJECT MANAGEMENT SUPPORT (CON'T)

	RISK ANALYSIS	COMPLEXITY ASSESSMENT	INITIAL COST ESTIMATES	COST-TO-COMPLETE PROJECTION	INCREMENTAL DATA ABOUT EXPENDITURES	PROJECT TRACKING	OTHER
OBJECTORY							
OOA	G	G				G	
OOA/ST							
OOD	G	G	R	R	R	G	
PAISley							
PDL/81		S*					
PRIDE	G	S*	S*	S*	S*	S*	*
PROMOD	G	S*				G	
PROTOB							
PSL/PSA	G*	G*	G*	G*	G*	G*	
RM		G				G	
SADT	G	G	G	G	G	G	
SCR							
SD		S					
SEM	G	G	S	G	G	G	
SSD							
STATEMATE	G	G				G	S*
StP		G*					
STRADIS	S	S	S	S	S	S	
TAGS		R	G	R	G	G*	
UCRA	S		G	G			
WARD/MELL							

S = SPECIFIC DIRECTIONS/PROCEDURES PRESCRIBED
G = GUIDELINES/FRAMEWORK PROVIDED
R = REQUIRED; NO DIRECTIONS PROVIDED
** = AUTOMATED SUPPORT PROVIDED*

890124-026M9-5

TABLE 13B. PROJECT MANAGEMENT SUPPORT

	PROJECT PLANNING	SCHEDULING/ MANPOWER LOADING	PERT/GANTT CHART GENERATION	PERSONNEL ALLOCATION	DEVELOPMENT RESOURCE ALLOCATION	CONFIGURATION MANAGEMENT	RELIABILITY ESTIMATES	OTHER
ABDLSLCM	G*	G*	G	G*	G	G*	G*	G*
ADM	S	S		S		G		
AISLE						R		
BOX STRUC	G	G	G	G	G	G		
BYRON PDL								
CORE								
DARTS								
DBO	S	S			S	S	S	S
DCDS	S*	S*				S*		
DSSAD								
ESA	S	S	S	S	S*	R	R	
E-DEV	S	S	S	S	S	R	R	
GYPSY	G						G	
HOOD				G	G	R		
IBM/4LDM	G							
IEM	S			G	G			S
ISAC	G	S	G	G	G	S	G	
ISTAR	S*	S*	S*	S*	S*	S*	*	
JSD								
MASCOT	G	G		G	G	R	S	
MBOOD								
MINI-ASYST	G	G		G	G	G	G	G
MULTI/CAM	S	S	S	S	S	S	S	

S = SPECIFIC DIRECTIONS/PROCEDURES PRESCRIBED
G = GUIDELINES/FRAMEWORK PROVIDED
R = REQUIRED; NO DIRECTIONS PROVIDED
** = AUTOMATED SUPPORT PROVIDED*

890124-027M9-5

TABLE 13B. PROJECT MANAGEMENT SUPPORT (CON'T)

	PROJECT PLANNING	SCHEDULING/ MANPOWER LOADING	PERT/GANTT CHART GENERATION	PERSONNEL ALLOCATION	DEVELOPMENT RESOURCE ALLOCATION	CONFIGURATION MANAGEMENT	RELIABILITY ESTIMATES	OTHER
OBJECTORY								
OOA	G					R		
OOA/ST								
OOD	R	R	N	R	R	G	R	
PAISley								
PDL/81								
PRIDE	S*	S*	S*	S*	S*	G	S*	S*
PROMOD		G		G	G	S	G	
PROTOB								
PSL/PSA	G*	G*		G*	G*	G*		
RM				G	G			
SADT	S	G	G	G	G	G		
SCR	R	R		R	R	R		
SD								
SEM	S	R		R	R	R		
SSD								
STATEMATE	G					S*	S*	
StP						G*		
STRADIS	S	S	G	S	G	S		
TAGS	R	R	R	R	R	S*	R	R
UCRA	S	S	G	G		R		
WARD/MELL								

S = SPECIFIC DIRECTIONS/PROCEDURES PRESCRIBED
G = GUIDELINES/FRAMEWORK PROVIDED
R = REQUIRED; NO DIRECTIONS PROVIDED
** = AUTOMATED SUPPORT PROVIDED*

890124-028M9-5

This page is intentionally blank.

6.14 RECORDING DECISIONS AND CHANGES

During the software process many decisions are made, and changes in requirements, specification, and design occur. One essential component for successful software development involves the recording of such decisions and changes. These records are frequently a key component in determining the maintainability of a system.

In question 22 of the survey, information was sought about the support provided by a method relative to maintaining a traceable record of technical decision-making during the software development process. Five recording activities were listed in the questionnaire. They were:

Records are maintained of specification and design options which were considered.

Records are maintained of any trade-off studies.

Records are maintained of the rationale for any decisions.

Records are maintained of the personnel who were involved in making a decision.

Records are maintained of all changes related to specification and/or design decisions.

Developers were asked to select the one response which best reflects the level of support provided by the method relative to maintaining the record. The following choice of responses was provided:

A: The method provides automated recording procedures.

S: The method provides specific directions for recording information.

R: The method requires that a record be kept, but does not provide specific directions for recording the information.

N: The method does not specifically require that a record be maintained.

A summary of the responses is provided in Table 14. A blank entry appears in the table if the developer specified that the method does not specifically require that a record be maintained, or if the developer did not respond for the particular activity.

TABLE 14. RECORDING DECISIONS AND CHANGES

	A RECORD IS MAINTAINED OF:				
	SPECIFICATION/DESIGN OPTIONS THAT WERE CONSIDERED	ANY TRADEOFF STUDIES	THE RATIONALE FOR ANY DECISION	THE PERSONNEL WHO WERE INVOLVED IN DECISION-MAKING	ALL CHANGES RELATED TO SPECIFICATION/ DESIGN DECISIONS
ABDLSLCM	R	R	R	R	R
ADM	R			R	R
AISLE	A	A	A	A	A
BOX STRUC	S	S	S	S	S
BYRON PDL					
CORE		S	S	S	
DARTS	R	R	R	R	R
DBO	AS	AS	AS	AS	AS
DCDS	A	A	A	A	A
DSSAD			R		R
ESA					
E-DEV	R	R	R	R	R
GYPSY					
HOOD					
IBM/4LDM					
IEM					
ISAC	A	S	S	R	R
ISTAR	A			A	A
JSD					
MASCOT	R	R	S		
MBOOD					
MINI-ASYST	R	R	R	R	R
MULTI/CAM	A	A	A	A	A

A = AUTOMATED PROCEDURES PROVIDED
S = SPECIFIC DIRECTIONS PROVIDED
R = REQUIRED; NO DIRECTIONS PROVIDED

890124-029M9-5

TABLE 14. RECORDING DECISIONS AND CHANGES (CON'T)

	A RECORD IS MAINTAINED OF:				
	SPECIFICATION/DESIGN OPTIONS THAT WERE CONSIDERED	ANY TRADEOFF STUDIES	THE RATIONALE FOR ANY DECISION	THE PERSONNEL WHO WERE INVOLVED IN DECISION-MAKING	ALL CHANGES RELATED TO SPECIFICATION/ DESIGN DECISIONS
OBJECTORY	A	A	A	A	A
OOA	S	S	S		R
OOA/ST					
OOD			R		R
PAISley					
PDL/81					
PRIDE	S	S	S	S	S
PROMOD	S	S	S	S	S
PROTOB					
PSL/PSA	A	S	S	S	S
RM					
SADT	S	S	S	S	S
SCR	R	R	R	R	R
SD					
SEM	S	S	S	S	
SSD					
STATEMATE					
StP					
STRADIS	S	S	S	S	S
TAGS	A	A	S	S	A
UCRA					
WARD/MELL					

A = AUTOMATED PROCEDURES PROVIDED
S = SPECIFIC DIRECTIONS PROVIDED
R = REQUIRED; NO DIRECTIONS PROVIDED

890124-030M9-5

This page is intentionally blank.

6.15 TESTING SUPPORT PROVIDED BY THE METHOD

Quality assurance, verification and validation, and testing are terms associated with an essential component of the software process, namely, that of assuring that the developing software product conforms to the system requirements. In question 21, developers were asked to describe the level of support that the method provides for assessing this conformance. The following testing activities were listed in the question:

Test planning at one or more precise points in the software process.

Generation of test based on system requirements.

Unit and integration testing.

Field testing and acceptance testing.

Generation of test data.

Regression testing.

Prescriptive checking of interfaces.

For each of the above activities, the resţ ondents were asked to select one of the following responses to indicate the level at which the method addresses the activity:

S: The method prescribes specific directions and procedures for conducting the activity.

G: The method provides guidelines or a framework for the activity, e.g., a document template.

R: The method requires the activity but does not provide directions for accomplishing the activity.

N: The method does not address the activity.

Developers were also asked to indicate if automated support for the particular testing activity is available from the method vendor. The responses for this question are summarized in Table 15. An asterisk is appended to the appropriate code when automated support is available. The developer was also given the opportunity to list other activities associated with testing. When the entry occurs under the column headed "Other", the reader is referred to the individual write-up for additional information. When the developer responded that the method does not address the activity, or when no response was provided, a blank entry occurs in the table.

The reader should note that activities associated with testing may not be the concern of a particular method. This is likely to be the case when the method addresses activities found in the early phases of development, such as requirement analysis or system specification. Even those method which focus on design may choose to consider the issue of testing as one which is distinct from that of design, and which should be addressed independently. The reader can consult Table 1 to see which activities of the software process are addressed by the method in order to determine whether the issue of testing is appropriate. The reader should also review the individual method write-ups to acquire further insight relative to goals of the method.

TABLE 15. TESTING SUPPORT

	TEST PLANNING AT PRECISE POINTS	TEST GENERATION BASED ON REQUIREMENTS	UNIT/INTEGRATION TESTING	FIELD TESTING/ ACCEPTANCE TESTING	GENERATION OF TEST DATA	REGRESSION TESTING	PRESCRIPTIVE CHECKING OF INTERFACES	OTHER
ABDLSLCM	G*	G*	G*	G	G	G	S*	
ADM	S		S		G	G	G	
AISLE	S	R	S*	S*	S*			
BOX STRUC	S	R	R	R	R	R	G	
BYRON PDL								
CORE								
DARTS	R	R	R	R	R	R	R	
DBO	S	S						S
DCDS	S*	S*	G	G*	R	G*	G*	
DSSAD	R	R	R					
ESA	R	R	R	R	R		R	S*
E-DEV	R	R	R	R	R		R	
GYPSY								S*
HOOD		G	G				G*	
IBM/4LDM								
IEM	S		S	G			S	
ISAC	G	G	G	G	R		G	
ISTAR								
JSD								
MASCOT	S	S	S	G	G	R	G	
MBOOD			R				G	
MINI-ASYST	G	G	G	G	G		G	
MULTI/CAM	S*	S	S	S	S	S	S	

S = SPECIFIC DIRECTIONS/PROCEDURES PRESCRIBED
G = GUIDELINES/FRAMEWORK PROVIDED
R = REQUIRED; NO DIRECTIONS PROVIDED
** = AUTOMATED SUPPORT PROVIDED*

890124-031M9-5

TABLE 15. TESTING SUPPORT (CON'T)

	TEST PLANNING AT PRECISE POINTS	TEST GENERATION BASED ON REQUIREMENTS	UNIT/INTEGRATION TESTING	FIELD TESTING/ ACCEPTANCE TESTING	GENERATION OF TEST DATA	REGRESSION TESTING	PRESCRIPTIVE CHECKING OF INTERFACES	OTHER
OBJECTORY	S*	S*	S*	S*	G			S*
OOA	G	R						
OOA/ST								
OOD	G	R	G	G	R	G	S	
PAISley	G	G					G	
PDL/81								
PRIDE	G	G	G	G	G		G	
PROMOD	G	G						
PROTOB								
PSL/PSA	G*	G*	G*		G*		G*	
RM	S							
SADT	G	G						
SCR	R	G	G	R	G		G	N
SD			R				R	
SEM	G	G	R	G	G	R	G	
SSD								
STATEMATE	S*	S*	G*	G*	G*		S*	S*
SIP								
STRADIS	S	G	S	S	S	S	G	
TAGS	G	G	G	G	G	G	G*	
UCRA								S
WARD/MELL		G						

S = SPECIFIC DIRECTIONS/PROCEDURES PRESCRIBED
G = GUIDELINES/FRAMEWORK PROVIDED
R = REQUIRED; NO DIRECTIONS PROVIDED
** = AUTOMATED SUPPORT PROVIDED*

890124-032M9-5

This page is intentionally blank.

6.16 DOCUMENTATION REQUIRED BY THE METHOD

In question 20 of the survey, information was sought regarding the documentation which is required to be produced when using the method. In addition to indicating what kinds of documents are required by the method, developers were asked to indicate the type of format required for various kinds of documentation. The following choices for responses were provided:

F: A fixed document format is prescribed by the method.

T: The document format is tailorable within the method.

N: The method requires the document, but does not prescribe the format.

Information was also requested regarding what automated support, if any, is provided with the method. Developers specified one of two choices when appropriate: either the document is automatically generated based on data produced from other steps in the use of the method, or the document is produced from user responses to computer-directed prompts.

The following list of documents was specified in the questionnaire: requirements definition, functional specification, behavioral specification, architectural specification, interface specification, system structure chart, data dictionary, design document, internal program documentation, quality assurance/test plan, support/installation plan, user manual, design decision log, problem log, and change log. Developers were also given the opportunity to list other kinds of documentation.

Tables 16A and 16B summarize the responses to the question. A code of F, T, or N indicates that the document is required to be produced when using the method, and also indicates the type of format provided. An additional entry of either G or P indicates the kind of automated support which is provided when using the method.

The results provided in these two tables are indicative of the coverage provided by the method since many of the documents in the above list are associated with specific activities in the software process. Accordingly, the reader is invited to compare the results listed in Tables 16A and 16B with the results provided in Table 1.

TABLE 16A. DOCUMENTATION

	REQUIREMENTS DEFINITION	FUNCTIONAL SPECIFICATION	BEHAVIORAL SPECIFICATION	ARCHITECTURAL SPECIFICATION	INTERFACE SPECIFICATION	SYSTEM STRUCTURE CHART	DATA DICTIONARY	DESIGN DOCUMENT
ABDLSLCM	TG	TG	TG	TG	TG	TG	TG	TG
ADM	N	N	N	N				N
AISLE	TG	TG	TG	FG	N	FG	FG	TG
BOX STRUC	TG	TG	TG	TG	TG	TG	TG	TG
BYRON PDL							TG	TG
CORE								
DARTS	N	T	F	T	T	F	T	T
DBO	TP	TP		TP	TP	TG		TG
DCDS	TG	TG	TG	TG	TG	TP	FG	TG
DSSAD	N	N	T	N	T	T	T	T
ESA	T	T	T	T	T	T	T	T
E-DEV	T	T	T	T	T	T	T	T
GYPSY		T						
HOOD				F	F		T	TG
IBM/4LDM		T		T	T	T	T	T
IEM	T	T	N	T	T	T	N	T
ISAC	FP	FP	T	T	FP	F	T	T
ISTAR								
JSD	N	T	FP	T	FG	F	FG	FG
MASCOT	N	T	T	F	F	FG	T	T
MBOOD	T	T	F	T	T	T	T	T
MINI-ASYST	T	T	T	T	T	T	N	T
MULTI/CAM	TG	TG	TG	TG	TG	TG	TG	TG

F = FIXED FORMAT
T = TAILORABLE FORMAT
N = REQUIRED; FORMAT NOT PRESCRIBED
G = GENERATED AUTOMATICALLY BASED ON DATA PRODUCED FROM OTHER STEP(S) IN THE METHOD
P = PRODUCED AUTOMATICALLY FROM USER RESPONSES TO COMPUTER-DIRECTED PROMPTS

890124-033M9-5

TABLE 16A. DOCUMENTATION (CON'T)

	REQUIREMENTS DEFINITION	FUNCTIONAL SPECIFICATION	BEHAVIORAL SPECIFICATION	ARCHITECTURAL SPECIFICATION	INTERFACE SPECIFICATION	SYSTEM STRUCTURE CHART	DATA DICTIONARY	DESIGN DOCUMENT
OBJECTORY	TP		TP	TG	TP	TG		TG
OOA	TP	TG	TG	T	T		T	
OOA/ST	T	T	T				T	
OOD	N	N	F	F	F	F	N	T
PAISley	F	F	F	F				
PDL/81	T	T						T
PRIDE	FG	TP	NP	FG	FG	FG	FG	FG
PROMOD	TG	TG	TG	TG	TG	TG	TG	TG
PROTOB		FP	FP					
PSL/PSA	TG	TG	TG	TG	TG	TG	TG	TG
RM						FG		FG
SADT	FG	FG	FG	FG	FG	FG	N	FG
SCR	T				T			T
SD				F	F	F		F
SEM	TP	FG	FG	TP	FG		TP	TP
SSD	T	T		T	T	T	T	T
STATEMATE		FG	FG	FG	FG	FG	FG	
StP	TG	TG		TG	TG	TG	TG	TG
STRADIS	TP	N	NP	N	NP	F	TP	TP
TAGS	FP	FP	FP	FP	FP	NP	FG	FP
UCRA	T	T					T	
WARD/MELL								

F = FIXED FORMAT

T = TAILORABLE FORMAT

N = REQUIRED; FORMAT NOT PRESCRIBED

G = GENERATED AUTOMATICALLY BASED ON DATA PRODUCED FROM OTHER STEP(S) IN THE METHOD

P = PRODUCED AUTOMATICALLY FROM USER RESPONSES TO COMPUTER-DIRECTED PROMPTS

890124-034M9-5

TABLE 16B. DOCUMENTATION

	INTERNAL PROGRAM DOCUMENTATION	QUALITY ASSURANCE/ TEST PLAN	SUPPORT/ INSTALLATION PLAN	USER MANUAL	DESIGN DECISION LOG	PROBLEM LOG	CHANGE LOG
ABDLSLCM	TG	TP	TP	TP	TP	TG	TG
ADM							
AISLE	TG	FG					
BOX STRUC	TG	TG	TG	TG	TG	TG	TG
BYRON PDL	F			TG			
CORE							
DARTS	T	N	N	N	N	N	N
DBO		T	N	N	FG	TG	TG
DCDS		N			TG	TG	TG
DSSAD	T	N	N	N			
ESA	T						
E-DEV	T	N	N	N	N	N	N
GYPSY							
HOOD	T		T				G
IBM/4LDM							
IEM	G	N	N	N	N		
ISAC	N	N	N	N	N	F	F
ISTAR		FP		TP		FG	FG
JSD	F						
MASCOT	N	F	N	N	N	N	N
MBOOD	N	N	N	N	N	N	N
MINI-ASYST	T	N	N	N	T	T	T
MULTI/CAM	TG	TG	TG	TG	TG	TG	TG

F = FIXED FORMAT
T = TAILORABLE FORMAT
N = REQUIRED; FORMAT NOT PRESCRIBED
G = GENERATED AUTOMATICALLY BASED ON DATA PRODUCED FROM OTHER STEP(S) IN THE METHOD
P = PRODUCED AUTOMATICALLY FROM USER RESPONSES TO COMPUTER-DIRECTED PROMPTS

890124-035M9-5

TABLE 16B. DOCUMENTATION (CON'T)

	INTERNAL PROGRAM DOCUMENTATION	QUALITY ASSURANCE/ TEST PLAN	SUPPORT/ INSTALLATION PLAN	USER MANUAL	DESIGN DECISION LOG	PROBLEM LOG	CHANGE LOG
OBJECTORY	FG	FG					FG
OOA							
OOA/ST							
OOD	T	N	N	N	N	N	N
PAISley							
PDL/81	T			T			
PRIDE	TG	TG	TG	FG	FG	NP	NP
PROMOD	TG	TG	N	N	TG	N	TG
PROTOB							
PSL/PSA	TG	TG		T	TG		TG
RM	FG				T		
SADT	FG	FG	FG				
SCR	T	T		N		T	T
SD							
SEM							
SSD							
STATEMATE		T					
StP							
STRADIS	N	TP	T	N	N	T	TP
TAGS	TG	NP	NP	NP	FP	FG	G
UCRA							
WARD/MELL							

890124-036M9-5

F = FIXED FORMAT
T = TAILORABLE FORMAT
N = REQUIRED; FORMAT NOT PRESCRIBED
G = GENERATED AUTOMATICALLY BASED ON DATA PRODUCED FROM OTHER STEP(S) IN THE METHOD
P = PRODUCED AUTOMATICALLY FROM USER RESPONSES TO COMPUTER-DIRECTED PROMPTS

This page is intentionally blank.

6.17 QUALIFICATIONS NEEDED FOR USE OF THE METHOD

The successful use of a method in developing software will depend on a number of factors. One of the most significant factors involves how well the method is suited to the development team members who ultimately are the ones to use the method. Question 33 of the survey was designed to solicit information relative to qualifications needed to successfully use a method. In order to provide a specific focus, the question asked about the minimum qualifications needed by a development team leader. The developer was asked to express his or her opinion on each of the following:

The minimum college-level technical education.

The minimum number of years of development experience.

The number of programming languages for which the person has a working knowledge.

The number of different software systems with which the person has had development, programming, and/or maintenance experience.

The responses have been summarized in Table 17. The reader is cautioned that it may have been difficult for the respondent to distinguish between the minimum qualifications needed to use the method, and the minimum qualifications needed to be a team leader. In electing to focus on the team leader, the authors recognize that the team leader will be the major factor in the correct employment of the method in the development process.

An examination of the results shows, in general, that emphasis is placed on the number of years of technical education, and on development experience. A relatively lesser importance is placed on knowledge of a variety of programming languages or on experience with a variety of systems.

In addition to question 33, information on the background needed for successful use of the method was sought in question 34. Here developers were asked their opinions regarding the major theoretical constructs that should be understood by an experienced developer for successful use of a particular method. The responses to this question have been given in the Technology Insertion section of the individual write-ups.

TABLE 17. TEAM LEADER QUALIFICATIONS NEEDED FOR METHOD USE

	YEARS OF TECHNICAL EDUCATION	YEARS OF DEVELOPMENT EXPERIENCE	NUMBER OF PROGRAMMING LANGUAGES KNOWN	NUMBER OF SYSTEMS WORKED WITH
ABDLSLCM	4	3 - 5	2	2
ADM	4	1 - 2	1	1
AISLE	4	0	1	1
BOX STRUC	4	1 - 2	2	3 - 4
BYRON PDL	2 - 3	0	1	1
CORE	< 2	1 - 2	0	2
DARTS	4	3 - 5	2	2
DBO	< 2	3 - 5	0	0
DCDS	4	3 - 5	2	3 - 4
DSSAD	2 - 4	0	1	1
ESA	4	3 - 5	1	1
E-DEV	4	3 - 5	1	1
GYPSY	4	1 - 2	2	2
HOOD	2 - 3	1 - 2	2	3 - 4
IBM/4LDM	4	3 - 5	2	2
IEM	2 - 3	3 - 5	0	1
ISAC	2 - 3	3 - 5	1	1
ISTAR	4	3 - 5	2	2
JSD	2 - 3	3 - 5	1	1
MASCOT	< 2	1 - 2	1	2
MBOOD	< 2	3 - 5	1	1
MINI-ASYST	2 - 3	1 - 2	1	1
MULTI/CAM	2 - 3	3 - 5	1	3 - 4
OOA	4	3 - 5	2	2
OOA/ST				
OOD	4	3 - 5	2	3 - 4
PAISley	4	3 - 5	2	2
PDL/81	2 - 3	1 - 2	1	1
PRIDE	< 2	0	1	1
PROMOD	4	3 - 5	2	2
PROTOB	4	0	1	1
PSL/PSA	4	1 - 2	1	1
RM	4	0	1	1
SADT	< 2	3 - 5	1	1
SCR	4 - 5	3 - 5	2	2
SD	< 2	1 - 2	1	2
OBJECTORY	4 - 5	3 - 5	2	1
SEM	4	3 - 5	1	1
SSD	4	1 - 2	0	0
STATEMATE	4	0	1	1
StP	4	3 - 5	2	2
STRADIS	< 2	1 - 2	1	1
TAGS	4	1 - 2	1	1
UCRA	4	3 - 5	0	3 - 4
WARD/MELL	4	3 - 5	1	2

890124-037M9-5

6.18 ASSISTANCE AVAILABLE FOR TRAINING IN THE USE OF THE METHOD

Table 18 presents a summary of the types of assistance that are available to provide an organization with training in the use of the method. The table is based upon the responses to question 25 of the survey which listed the following types of assistance:

Hands-on demonstrations.

Overview presentations and/or classroom tutorials.

On-site consulting by the vendor or by independent consultants.

On-line tutorials and/or tutorials on video tape.

An on-line help facility.

A "hot-line" service.

User manuals/documentation for the method.

Users' support group.

Related publications from third parties.

Periodic technical updates, e.g., newsletters or bulie'ins.

Information relative to the cost of training was sought in question 29. Responses for this latter question can be found in the Acquisition Factors section of the individual write-ups.

TABLE 18. ASSISTANCE AVAILABLE TO TRAIN ORGANIZATION IN THE USE OF THE METHOD

	HANDS-ON DEMONSTRATIONS	OVERVIEW PRESENTATIONS AND/OR CLASSROOM TUTORIALS	ON-SITE CONSULTING BY THE VENDOR OR INDEPENDENT CONSULTANTS	ON-LINE TUTORIALS AND/OR VIDEO TAPES	ON-LINE HELP FACILITY	A "HOT-LINE" SERVICE	USER MANUALS	USERS' SUPPORT GROUP	RELATED PUBLICATIONS FROM THIRD PARTIES	PERIODIC TECHNICAL UPDATES (e.g., NEWSLETTERS, BULLETINS)
ABDLSLCM		●	●			●	●			●
ADM		●	●							
AISLE	●	●	●		●	●	●	●	●	·
BOX STRUC	●	●	●	●					●	●
BYRON PDL			●	●	●	●	●	●		●
CORE	●	●	●		●	●	●			
DARTS		●	●							
DBO	●	●	●	●	●		●		●	●
DCDS		●	●		●	●	●			●
DSSAD		●	●				●			
ESA	●	●	●				●	●	●	●
E-DEV	●	●	●				●	●	●	●
GYPSY	●	●	●		●		●	●		
HOOD	●	●	●				●	●		
IBM/4LDM		●								
IEM	●	●	●	●		●	●	●	●	●
ISAC	●	●	●			●	●	●		
ISTAR	●	●	●		●	●	●	●		●
JSD	●	●	●				●	●		
MASCOT		●	●			●		●		●
MBOOD		●	●							
MINI-ASYST	●	●	●			●	●			●
MULTI/ CAM	●	●	●	●	●	●	●	●	●	●

890124-038M9-5

TABLE 18. ASSISTANCE AVAILABLE TO TRAIN ORGANIZATION IN THE USE OF THE METHOD (CON'T)

	HANDS-ON DEMONSTRATIONS	OVERVIEW PRESENTATIONS AND/OR CLASSROOM TUTORIALS	ON-SITE CONSULTING BY THE VENDOR OR INDEPENDENT CONSULTANTS	ON-LINE TUTORIALS AND/OR VIDEO TAPES	ON-LINE HELP FACILITY	A "HOT-LINE" SERVICE	USER MANUALS	USERS' SUPPORT GROUP	RELATED PUBLICATIONS FROM THIRD PARTIES	PERIODIC TECHNICAL UPDATES (e.g., NEWSLETTERS, BULLETINS)
OBJECTORY		●	●				●			
OOA		●	●					●	●	●
OOA/ST		●	●						●	
OOD	●	●	●	●			●		●	
PAISley	●	●				●	●		●	
PDL/81			●			●	●			
PRIDE	●	●	●		●	●	●			●
PROMOD	●	●	●	●	●	●	●	●	●	●
PROTOB	●	●	●		●		●			
PSL/PSA	●	●	●	●	●	●	●	●	●	●
RM	●	●	●							
SADT		●	●				●		●	
SCR	●	●		●			●		●	●
SD		●	●						●	
SEM		●	●				●		●	
SSD	●	●	●	●						
STATEMATE	●	●	●	●		●	●	●	●	●
StP	●	●	●		●	●	●	●	●	●
STRADIS		●	●			●	●	●		●
TAGS	●	●	●	●		●	●			
UCRA		●	●				●			
WARD/MELL	●	●	●		●	●	●	●		

890124-039M9-5

This page is intentionally blank.

6.19 ESTIMATED LEARNING TIMES FOR THE METHOD

In questions 26, 27 and 28, the developers were asked to express their opinions relative to the following questions associated with the amount of time needed to learn to use a method:

How many days would be required for a project manager to acquire an understanding of the major features and benefits of the method?

How many days would be required for an experienced developer (five or more years' practice) to learn to use the essentials of the method?

How many months would be required for an experienced developer to achieve the level of expert user of the method?

For each of the questions, the responses were analyzed and grouped into three categories in order to facilitate a comparison of the methods. The results are presented in Table 19. The actual estimates of the developers have been provided in the Technology Insertion section of the individual write-ups.

The authors have observed that there may be a tendency of developers to equate learning time with the amount of time provided in the typical training course. Past experience with surveying users of methods relative to the same questions has shown that the users often provide higher estimates for the learning times.

It contrasting the methods relative to the issue of learning times, it is advisable to also consult Table 17. In this latter table, the developers have provided some idea of the technical background which is needed to successfully use the method. It is obvious that the technical background of a person will influence the amount of time needed by the person to learn a method. Additionally, one can expect that the quality and type of training available will also have an effect on the learning time.

TABLE 19. ESTIMATED LEARNING TIMES FOR METHOD

	NUMBER OF DAYS NEEDED FOR PROJECT MANAGER TO LEARN BASICS OF METHOD			NUMBER OF DAYS NEEDED FOR DEVELOPER TO LEARN TO USE METHOD			NUMBER OF MONTHS NEEDED FOR EXPERIENCED DEVELOPER TO ACHIEVE LEVEL OF EXPERT USER		
	2 OR LESS	3 OR 4	5 OR MORE	3 OR LESS	4 OR 5	6 OR MORE	2 OR LESS	3 TO 5	6 OR MORE
ABDLSLCM	●					●	●		
ADM	●				●			●	
AISLE	●				●		●		
BOX STRUC			●			●			●
BYRON PDL	●				●		●		
CORE	●				●				●
DARTS	●			●				●	
DBO	●				●		●		
DCDS		●			●		●		
DSSAD			●			●	●		
ESA			●			●			●
E-DEV		●				●			●
GYPSY	●				●			●	
HOOD		●				●		●	
IBM/4LDM	●			●					●
IEM		●				●			●
ISAC	●			●				●	
ISTAR		●				●			●
JSD			●			●			●
MASCOT	●				●			●	
MBOOD	●					●	●		
MINI-ASYST	●					●		●	
MULTI/CAM			●	●			●		

890124-040M9-5

TABLE 19. ESTIMATED LEARNING TIMES FOR METHOD (CON'T)

	NUMBER OF DAYS NEEDED FOR PROJECT MANAGER TO LEARN BASICS OF METHOD			NUMBER OF DAYS NEEDED FOR DEVELOPER TO LEARN TO USE METHOD			NUMBER OF MONTHS NEEDED FOR EXPERIENCED DEVELOPER TO ACHIEVE LEVEL OF EXPERT USER		
	2 OR LESS	3 OR 4	5 OR MORE	3 OR LESS	4 OR 5	6 OR MORE	2 OR LESS	3 TO 5	6 OR MORE
OBJECTORY	●					●		●	
OOA			●			●		●	
OOA/ST	●				●				●
OOD		●			●				●
PAISley		●				●		●	
PDL/81	●			●			●		
PRIDE	●				●				●
PROMOD	●				●		●		
PROTOB	●					●	●		
PSL/PSA	●					●		●	
RM			●			●		●	
SADT	●				●		●		
SCR		●				●		●	
SD	●				●				●
SEM	●					●		●	
SSD	●				●		●		
STATEMATE	●			●				●	
StP	●			●				●	
STRADIS			●			●			●
TAGS	●				●		●		
UCRA	●			●				●	
WARD/MELL	●					●			●

890124-041M9-5

This page is intentionally blank.

APPENDIX A

BIBLIOGRAPHY

GENERAL INFORMATION ON METHODOLOGY

[Abbo83] R. J. Abbott, "Program Design by Informal English Descriptions", Communications of the ACM, Vol. 26, Nov. 1983.

[Abbo85] R. Abbott, An Integrated Approach to Software Development. New York, NY: John Wiley and Sons, Inc., 1985.

[Agre86] W. W. Agresti, Tutorial: New Paradigms for Software Development. Los Angeles, CA: The Computer Society Press, 1986.

[Ague87] U. Aguero and S. Dasgutpa, "A Plausibility Driven Approach to Computer Architecture Design", Communications of the ACM, Vol. 30, No. 8, Aug. 1987.

[Albr83] A. J. Albrecht and J. Gaffney, Jr., "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation", IEEE Transactions on Software Engineering, Volume 9, No. 6, November 1983, pp. 639-648.

[Bake72] F. Baker, "Chief-Programmer Team Management of Production Programming", IBM Systems Journal, Vol. 1, 1972.

[Bake86] T. P. Baker and G. M. Scallon, "An Architecture for Real-Time Software Systems", IEEE Software, Vol. 3, No. 3, May 1986.

[Blan83] J. Blank, M. Drummen, H. Gersteling, T. Janssen, M. Krijger, and W. Pelger, Software Engineering: Methods and Techniques. New York, NY: John Wiley & Sons, 1983.

[Berg81a] G. D. Bergland, "A Guided Tour of Program Design Methodologies", IEEE Computer, Vol. 14, Oct. 1981, pp. 13-37.

[Berg81b] G. D. Bergland and R. D. Gordon, Tutorial: Software Design Strategies. Los Angeles, CA: The Computer Society Press, 1981.

[Bers87] E. H. Bersoff, B. J. Gregor, and A. M. Davis, "A New Look at the C3I Software Life Cycle", Signal, April 1987.

[Blum82] B. Blum, "The Life Cycle -- A Debate over Alternate Models", SIGSOFT Software Engineering Notes, Vol. 7, Oct. 1982.

[Boeh81] B. Boehm, Software Engineering Economics. Englewood Cliffs, NJ: Prentice Hall, Inc., 1981.

GENERAL INFORMATION ON METHODOLOGY

[Boeh88] B. Boehm, "A Spiral Model of Software Development and Enhancement", IEEE Computer, Vol. 21, No. 5, May 1988.

[Broo75] F. Brooks, The Mythical Man Month. Reading, MA: Addison-Wesley, 1975.

[Bran81] M. Branstad and W. Adrion, "NBS Programming Environment Workshop Report," SIGSOFT Software Engineering Notes, Vol. 6, August 1981.

[Card87] D. N. Card, F. E. McGarry and G. T. Page, "Evaluating Software Engineering Technologies", IEEE Transactions on Software Engineering, Vol. SE-13, No. 7, July 1987.

[Carr86] M. A. Carrio, Jr., "The Technology Life Cycle and Ada", Defense Science and Electronics, Vol. 5, No. 7, July 1986.

[Cava87] J. Cavano, and F. LaMonica, "Quality Assurance in Future Development Methods," IEEE Software, Sept. 1987, pp. 26-34.

[Chat88] A. T. Chatfield and T. M. Rajkumar, "Comparison of Analysis Techniques for Information Requirement Determination", Communications of the ACM, Vol. 31, No. 9, Sep. 1988.

[Comp85] C'mputer, Issue on Visual Programming, Vol. 18, Aug. 1985.

[Conv85] R. Converse, C. W. McDonald, W. Riddle, and C. Youngblut, "Stars Methodology Area Summary, Vol. I: Organization and Plans", Institute for Defense Analyses, Alexandria, VA, Marcn 1985.

[Curt88] B. Curtis, H. Krasner, and N. Iscoe, "A Field Study of the Software Design Process for Large Systems", Communications of the ACM, Vol. 31, No. 11, Nov. 1988.

[Dart87] S. A. Dart, R. J. Ellison, P. H. Feiler and A. N. Haberman, "Software Development Environments", IEEE Computer, Vol. 20, No. 11, Nov. 1987, pp. 18-28.

[Davi87] B. David, "Costly Bugs", Wall St. Journal, Princeton, NJ, Jan. 28, 1987.

[Davi88b] A. Davis, E. Bersoff, and E. Comer, "A Strategy for Comparing Alternative Software Development Life Cycle Models," IEEE Transactions on Software Engineering, Vol. 14, No. 10, Oct. 1988, pp. 1453-1461.

[DeRe76] R. DeRemer and H. Kron, "Programming-in-the-large versus programming-in-the-small", IEEE Transactions on Software Engineering, Vo. SE-2, June 1976, pp. 80-86.

[Dijk69] E. W. Dijkstra, "Structured Programming," in Software Engineering Techniques, edited by J. N. Buxton and B. Randell. Rome: NATO Science Committee, 1969, pp. 88-93.

[Dijk72] E. W. Dijkstra, "Notes on Structured Programming," in Structured Programming, by O. J. Dahl, E. W. Dijkstra, and C. A. R. Hoare. New York: Academic Press, 1972, pp. 1-82.

[DInd81] Department of Industry, "Ada-based System Development Methodology Study Report", Vol. 1 and Anncx A, London, 1981.

GENERAL INFORMATION ON METHODOLOGY

[Druf83] L. Druffel, "Software Technology for Adaptable, Reliable Systems - Program Strategy," SIGSOFT Software Engineering Notes, Vol. 8, April 1983.

[Druf82] L. Druffel, "The Potential Effect of Ada on Software Engineering in the 1980's", SIGSOFT Software Engineering Notes, Vol. 7, July 1982.

[Earl86] M. Early, "Relating Software Requirements to Software Design", SIGSOFT Software Engineering Notes, Vol. 11, July 1986, pp. 37-39.

[Fair85] R. E. Fairley, Software Engineering Concepts. New York, NY: McGraw-Hill Book Co., 1985.

[Free82] P. Freeman and A. I. Wasserman, "Software Development Methodologies and Ada; Ada Methodologies Concepts and Requirements, Ada Methodology Questionnarie Summary, Comparing Software Design", University of California, Irvine, Ca., Nov. 1982.

[Free77] P. Freeman and A. I. Wasserman, Tutorial on Software Design Techniques. IEEE Computer Society, 1977.

[Garg87] A. Gargaro and T. L. Pappas, "Reusability Issues and Ada", IEEE Software, Vol. 4, No. 4, July 1987.

[Geha86] N. Gehani and A. McGettrick, Software Specification Techniques. Reading, MA: Addison-Wesley Publishing Co., 1986.

[Gerh86] M. S. Gerhardt, "Design Methodologies--Why Bother????", presentation at Ada Expo '86, Charleston, West Va., Nov. 1986.

[Guim85] T. Guimaraes, "A Study of Application Program Development Techniques", Communications of the ACM, Vol. 28, No. 5, May 1985, pp. 494-499.

[Hara88] M. Harandi, et al, "Report of the Fourth International Workshop on Software Specification and Design: Monterey, CA., April 3-4, 1987," SIGSOFT Software Engineering Notes, Vol. 13, No. 1, Jan. 1988, pp. 29-45.

[Hard87] M. Hardwick and D. L. Spooner, "Comparison of Some Data Models for Engineering Objects", IEEE Computer Graphics & Applications, March 1987.

[Hare88] D. Harel, "On Visual Formalisms", Communications of the ACM, Vol. 3 No. 5, May 1988.

[Henr85] S. M. Henry, J. D. Arthur, and R. E. Nance, "A Procedural Approach to Evaluating Software Development Methodologies," Technical Report TR-85-20, Virginia Polytechnic Institute, 30 March, 1985.

[Hind83] H. J. Hindin and W. B. Rauch-Hindin, "Special Series on System Integration," Electronic Design, Jan 6, 1983.

[Hoar87] C. A. R. Hoare, "An Overview of Some Formal Methods for Program Design", IEEE Computer, Vol. 20, No. 9, Sep. 1987.

GENERAL INFORMATION ON METHODOLOGY

[Houg87] R. C. Houghton, Jr., and D. R. Wallace, "Characteristics and Functions of Software Engineering Environments: an Overview", ACM SIGSOFT Software Engineering Notes, Vol. 12, No. 1, Jan. 1987.

[Hump88] W. Humphrey, "Characterizing the Software Process," IEEE Software, March 1988, pp. 73-79.

[ICSE87] Proceedings of the Ninth International Conference on Software Engineering, March 30-April 2, 1987, Monterey, CA, IEEE Catalog Number 87CH2432-3.

[IEEE83] IEEE Standard Glossary of Software Engineering Terminology, New York, NY, Feb. 1983.

[IEEE86] IEEE Transactions on Software Engineering, Special Issue on Software Design Methods, Vol. SE-12, Feb. 1986.

[IITR85] IIT Research Institute, "Draft Glossary of Software Quality Terms", Rome, N. Y., Nov. 1985.

[Isaa84] J. Isaak, Designing Systems for Real-time Applications," Byte, April 1984.

[IWSP85] SIGSOFT Software Engineering Notes, Special Issue - International Workshop on the Software Process and Software Environments, Vol. 11, Aug. 1986.

[Jens79] R. W. Jensen and C. C. Tonies, Software Engineering. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1979.

[John86] A. L. Johnson, "Software Engineering Combines Management and Technical Skills", SEI Bridge, Aug/Sept. 1986.

[Kell87] J. C. Kelly, "A Comparison of Four Design Methods for Real-Time Systems", Proceedings of the 9th International Software Engineering Conference, Monterrey, Ca., April 1987.

[Kell88] M. Kellner, and G. Hansen, "Software Process Modeling," (preliminary report), Software Engineering Institute, Pittsburgh, PA., May 1988.

[Keys86] Keystone Computer Association, Inc., "System Engineering Environment Tool Comparisons", Ft. Washington, Pa., Oct. 1986.

[King84] D. King, Current Practices in Software Development. New York, NY: Yourdon Press, 1984.

[Klei78] K. Kleine, "Selected Annotated Bibliography on Software Engineering", SIGSOFT Software Engineering Notes, Vol. 3, Jan. 1978.

[Ladd88] R. Ladden, "A Survey of Issues to be Considered in the Development of an Object-Oriented Development Methodology for Ada," SIGSOFT Software Engineering Notes, Vol. 13, No. 3, July 1988, pp. 24-31.

[Lefk82] D. Lefkovitz and H. Hill, "The Applicability of Software Development Methodologies to Naval Embedded Computer Systems", University of Pennsylvania, Moore School of Electrical Engineering, Philadelphia, Pa., November 9, 1982.

GENERAL INFORMATION ON METHODOLOGY

[Lehm84] M. Lehman, "A Further Model of Coherent Programming Processes", IEEE Proceedings: Software Process Workshop, Feb. 1984, pp. 27-33.

[Lehm85] M. M. Lehman and L. A. Belady, Program Evolution - Processes of Software Change. New York, NY: Academic Press, 1985.

[Liff85] B. Liffick, The Software Developer's Sourcebook. Reading, MA: Addison-Wesley Publishing Co., 1985.

[Ling79] R. Linger, H. Mills, and B. Witt, Structured Programming: Theory and Practice. Reading, MA: Addison-Wesley, 1979.

[Lisk86] B. Liskov and J. Guttag, Abstraction and Specifications in Program Development. New York, NY: MIT Press/McGraw-Hill, 1986.

[Maha87] L. Mahajan, M. Ginsberg, R. Pirchner, and R. Guilfoyle, Software Methodology Catalog, Report No. C01 001 C8 0001, Center for Software Engineering, Advanced Software Technology, U.S. Army CECOM, Fort Monmouth, NJ, Oct. 1987.

[Mars86] G. Marshall, Systems Analysis and Design: Alternate Structured Approaches. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1986.

[McDo86] C. W. McDonald, W. Riddle, and C. Youngblut, "STARS Methodology Area Summary," SIGSOFT Software Engineering Notes, Vol. 11, April 1986.

[Meye86] B. Meyer, "Genericity versus Inheritance", OOPSLA 1986 Conference Proceedings, special issue of SIGPLAN Notices, Vol. 21, No. 11, Nov. 1986, pg. 391-405.

[Mill80] H. D. Mills, "The Management of Software Engineering. Part I: Principles of Software Engineering", IBM Systems Journal, Vol. 13, No. 4, 1980.

[Mill83] H. D. Mills, Software Productivity. Little, Brown, and Co., 1983.

[Mill86] H. D. Mills, "Structured Programming: Retrospect and Prospect", IEEE Software, Vol. 3, Nov. 1986.

[Mill87] H. D. Mills, M. Dyer, and R. Linger, "Cleanroom Software Engineering", IEEE Software, Vol. 4, No. 5, Sep. 1987.

[Mill88] H. D. Mills, "Stepwise Refinement and Verification in Box-Structured Systems", IEEE Computer, Vol. 21, No. 6, June 1988.

[Misr88] S. Misra, and P. Jalics, "Third-Generation versus Fourth-Generation Software Development," IEEE Software, July 1988, pp. 8-14.

[Musa83] J. Musa, ed., "Stimulating Software Engineering Progress - A Report of the Software Engineering Planning Group," SIGSOFT Software Engineering Notes, April 1983.

[Niel87] K. W. Nielsen and K. Shumate, "Designing Large Real-Time Systems with Ada", Communications of the ACM, Vol. 30, No. 8, Aug. 1987.

GENERAL INFORMATION ON METHODOLOGY

[Neum81] P. Neuman and S. Walker, organizers, "Report - VERkshop II : Verification Workshop," SIGSOFT Software Engineering Notes, July 1981.

[Oste81] L. Osterweil, "Software Engineering Research: Directions for the Next Five Years", IEEE Computer, Vol. 14, April 1981.

[Park78] J. Parker, "A Comparison of Design Methodologies", SIGSOFT Software Engineering Notes, Oct., 1978.

[Pete81] L. J. Peters, Software Design: Methods and Techniques. New York, NY: Yourdon Press, 1981.

[Pfle87] S. L. Pfleeger, Software Engineering - The Production of Quality Software. New York, NY: Macmillan Publishing Co., 1987.

[Phil83] G. Philip, Software Design and Development. Science Research Associates, Inc., 1983.

[Pres87] R. S. Pressman, Software Engineering: A Practitioner's Approach, 2nd Ed. McGraw-Hill, Inc. 1987.

[Rama84] C. V. Ramamoorthy, A. Prakash, W. T. Tsai, and Y. Usuda, "Software Engineering: Problems and Perspectives", Computer, Vol. 17, Oct. 1984, pp. 191-209.

[Rama86] C. V. Ramamoorthy, V. Garg, and A. Parkash, "Programming in the Large," IEEE Transactions on Software Engineering, Vol. SE-12, No. 7, July 1886.

[Ross75] D. T. Ross, J. B. Goodenough, and C. A. Irvine, "Software Engineering: Process, Principles, and Goals", IEEE Computer, May 1975.

[RoyD84] D. M. Roy, "Software Tools and Methodology for NASA MSOCC Comprehensive Report", Century Computing, Laurel, MD, Sept. 84.

[Samm86] J. E. Sammet, "Why Ada is Not Just Another Programming Language," Communications of the ACM, Vol. 29, No. 8, Aug. 1986.

[SEIn85] SEI, "Software Factory Workshop: Preliminary Meetings", Software Engineering Institute, Pittsburgh, PA, Sept. 1985.

[Shaw86] M. Shaw, "Beyond Programming-in-the-Large: the Next Challenges for Software Engineering", Software Engineering Institute, Pittsburgh, PA, May 1986.

[SIGS82] SIGSOFT Software Engineering Notes, Report - Rapid Prototyping Workshop, Vol. 7, Dec. 1982.

[Smit86] C. U. Smith, R. C. Houghton, and C. R. Martin, "A Specification for a Methodology Consumer Guide (Final Report)", L & S Computer Technology, Inc., in cooperation with Duke University, Jan. 14, 1986.

[Soft83] Softfair 1983 panel, "Tools and Methodologies: The Perfect Match or the Odd Couple", sponsored by IEEE, Washington, DC, July 1983.

GENERAL INFORMATION ON METHODOLOGY

[Soft82] Softech, Inc., "Ada Software Design Methods Formulation, Case Studies Report, Contract DAAK80-80-C-0187, CENTACS, CECOM, Oct. 1982.

[Soft86] Softech, Inc., "Methods and Tools Technology Survey Interim Report; ISEC Methods and Tools Investigation", Waltham, MA, 31 March 86.

[Somm85] I. Sommerville, Software Engineering, 2nd ed. Reading, MA: Addison-Wesley, 1985.

[Srin87] A. Srinivasan and K. M. Kaiser, "Relationships Between Selected Organizational Factors and Systems Development", Communications of the ACM, Vol. 30, No. 6, June 1987.

[Stan88] J. A. Stankovich, "Misconceptions About Real-Time Computing: A Serious Problem for Next-Generation Systems", IEEE Computer, Vol. 21, No. 10, Oct. 1988.

[Sumr86] G. Sumrall, "Software Methodology", presentation to U.S. Army CECOM, 27 Aug 86.

[Syst83] Systems Designers Limited, "Life Cycle Support in the Ada Environment", Hampshire, England, March 1983.

[Webs88] D. E. Webster, "Mapping the Design Information Representation Terrain", IEEE Computer, Vol. 21, No. 12, Dec. 1988, pp. 8-23.

[Wegn84] P. Wegner, "Part 2: Programming in the Large," IEEE Software, Vol. 1, No. 3, July 1986.

[Wood88] B. Wood, R. Pethia, L Gold, and R. Firth, "A Guide to the Assessment of Software Development Methods," (preliminary report), Software Engineering Institute, April 1988.

[Yank88] N. Yankelovich, B. J. Haan, N. K. Meyrowitz and S. M. Drucker, "Intermedia: The Concept and the Construction of a Seamless Information Environment", IEEE Computer, Vol. 21, No. 1, Jan. 1988.

[YauS86] S. S. Yau and J. P. Tsai, "A Survey of Software Design Techniques," IEEE Transactions of Software Engineering, Vol. SE-12., No. 6, June 1986.

[YehR77] R. T. Yeh, ed., Current Trends in Programming Methodology. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1977.

[Your79a] E. Yourdon, ed., Classics in Software Engineering. New York, NY: Yourdon Press, 1979.

[Your82] E. Yourdon, ed., Writings of the Revolution. New York, NY: Yourdon Press, 1982.

INFORMATION ON SPECIFIC METHODS

[Alfo85] M. Alford, "SREM at the Age of Eight: the Distributed Computing Design System, " IEEE Computer, Vol. 18, April 1985, pp. 34-36.

[Alfo87a] M. Alford, "DCDS Multiple View Approach to Closing the Requirements/Design Gap ", presentation at the Fourth Conference on Methodologies and Tools for Real-Time Systems, Washington, DC, Sept. 14-15, 1987.

[Alfo87b] M. Alford, "Requirements Driven Test Planning", presentation at Software Testing and Validation Conference, Washington, DC, Sept. 23-24, 1987.

[Alfo88] M. Alford, "RDD Approach to Systems Engineering", Requirements Driven Developer, Vol. 1, No. 1, Dec. 1988.

[Auer00] Auerbach Publishers, Inc., "The ISAC Approach to User-Oriented Systems Specification", in Systems Development Management, Information Systems Analysis: Analysis Methods and Tools.

[Bail89] S. C. Bailin, "An Object-Oriented Requirements Specification Method", Communications of the ACM (to be published in the May 1989 issue).

[Bake86] T. Baker and G. Scallon, "An Architecture for Real-Time Software Systems", IEEE Software, May 1986, pp. 50-58.

[Bald88a] M. Baldessari and G. Bruno, "An Environment for Object-oriented Conceptual Programming Based on PROT Nets", in Advances in Petri Nets 1988, Springer-Verlag, 1988.

[Bald88b] M. Baldessari, V. Berti, and G. Bruno, "Object-oriented Conceptual Programming Based on PROT Nets", Proceedings of the International Conference on Computer Languages '88, Miami, FL, Oct. 1988.

[Balz83] R. Balzer, T. Cheatham, and C. Green, "Software Technology in the 1990's: Using the New Paradigm," IEEE Computer, Vol. 16, Nov. 1983, pp. 39-45.

[Barr87] B. M. Barry, D. A. Thomas, J. R. Altoft, M. Wilson, "Using Objects to Design and Build Radar ESM Systems", Proceedings of ACM OOPSLA '87, Orlando, FL, 1987.

[Bela76] L. Belady, and M. Lehman, "A Model of Large Program Development," IBM Systems Journal, Vol. 15, No. 3, 1976, pp. 225-252.

[Bjor87] D. Bjorner, "An Annotated VDM Bibliography", published by the Dansk Datamatik Center, Lyngby, Denmark, 1987.

[Blum87a] B. Blum, "The Tedium Development Environment for Information Systems", IEEE Software, Vol. 4, March 1987.

[Blum87b] B. I. Blum, "An Overview of TEDIUM", Information and Software Technology, Vol. 29, No. 9, Nov. 1987, pp. 475-496.

[Blum88] R. Blumofe, and A. Hecht, "Executing Real-Time Structured Analysis Specifications," SIGSOFT Software Engineering Notes, Vol.13, No. 3, July 1988, pp. 32-40.

INFORMATION ON SPECIFIC METHODS

[Boeh88] B. Boehm, "A Spiral Model of Software Development and Enhancement," IEEE Computer, Vol. 21, No. 5, May 1988, pp. 61-72.

[Booc82] G. Booch, Software Engineering with Ada. Menlo Park, CA: Benjamin/Cummings Publishing Co., 1982.

[Booc86] G. Booch, "Object-Oriented Development", IEEE Transactions on Software Development, Vol. SE-12, No. 2, Feb. 1986, pp. 211-221.

[Booc87] G. Booch, Software Components with Ada. Menlo Park, CA: Benjamin Cummings Publishing Co., 1987.

[Boyd87] S. Boyd, "Object-Oriented Design and PAMELA", SIGAda Ada Letters, Vol. 7, No. 4, July-Aug. 1987.

[Brun86a] G. Bruno and G. Marchetto, "Process-Translatable Petri Nets for the Rapid Prototyping of Process Control Systems", IEEE Transactions on Software Engineering, Vol. SE-12, Feb. 1986, pp. 346-357.

[Brun86b] G. Bruno, P. Spiller, and I. Tota, "AISPE: an Advanced Industrial Software Production Environment", Proceedings of IEEE Computer Software and Applications Conference (COMPSAC), Chicago, Oct. 1986, pp. 94-99.

[Brun86c] G. Bruno and A. Elia, "Operational Specification of Process Control Systems: Execution of PROT Nets Using OPS5", 10th World IFIP Congress, Dublin, Sept. 1986, pp. 35-40.

[Bruy88] W. Bruyn, R. Jensen, D. Keskar, and P. Ward, "ESML, An Extended Systems Modeling Language Based on the Data Flow Diagram," SIGSOFT Software Engineering Notes, Vol. 13, No. 1, Jan. 1988, pp. 58-67.

[Buhr84] R. J. A. Buhr, System Design with Ada. Englewood Cliffs, NJ: Prentice Hall, 1984.

[Buhr85a] R. J. A. Buhr, G. M. Karam, and C. M. Woodside, "An Overview and Example of Application of CAEDE: A New, Experimental Design Environment for Ada", presented at International Ada Conference, Paris, France, May 85.

[Buhr85b] R. J. A. Buhr, et al, "Experiments with Prolog Design Descriptions and Tools in CAEDE: an Iconic Design Environment for Multitasking, Embedded Systems", Dept. of Systems and Computer Engineering, Carleton Univ., Ottawa, Canada, Jan. 1985.

[Buhr88] R. J. A. Buhr, "Visual Prototyping in System Design", SCE Report 88-14, Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada, Sept. 14, 1988.

[Butl86] J. Butler, "Real-Time Systems: Methodologies and Tools", System Development, Nov. 1986.

[Cain75] S. H. Caine and E. K. Gordon, "PDL - A Tool for Software Design", Proceedings of the 1975 National Computer Conference, 1975, pp. 271-276.

[Cain88] "PDL/81 - An Introduction", October, 1988. Available from Caine, Farber & Gordon, Inc.

INFORMATION ON SPECIFIC METHODS

[Came83] J. R. Cameron, Tutorial: JSP & JSD: The Jackson Approach to Software Development. Los Angeles, CA: The Computer Society Press, 1983.

[Came86] J. R. Cameron, "An Overview of JSD", IEEE Transactions on Software Engineering, Vol. SE-12, No. 2, Feb. 1986.

[Cart88a] J. Carter, "MMAIM: A Software Development Method for Ada," SIGAda Ada Letters, Vol. 8, No. 3, May/June 1988, pp. 107-114.

[Cart88b] J. R. Carter, "MMAIM - A Software Development Method for Ada; Part II - Example", SIGAda Ada Letters, Sept./Oct. 1988.

[Case87] A. Case, Team System Analysis. Englewood Cliffs, NJ: Prentice-Hall, 1987.

[CCIT85a] Functional Specification and Description Language (SDL). Recommendations Z.100-Z.104 and Annexes, CCITT Red Book Vol. VI, Fasc.VI.10 and VI.11, Geneva, 1985.

[CCIT85b] CCITT High Level Language (CHILL). Recommendations Z.200, CCITT Red Book Vol.VI, Fasc.VI.12, Geneva, 1985.

[CCIT86] International Telegraph and Telephone Consultative Committee, "G/T - A Natural Evolution from SDL", June 1986, pp. 1-26, A.1-A.38. (internal report from ASA Ltd, UK to CCITT)

[Chao86] A. Chao, "A Modular Approach to Real-Time Software", Computer Design, Oct. 1986.

[Chat88] A. Chatfield, and T. Rejkumar, "Comparison of Analysis Techniques for Information Requirements Determination," Communications of the ACM, Vol. 31, No. 9, Sept. 1988, pp. 1090-1097.

[Chen76] P. P. Chen, "The Entity-Relationship Model - Toward a Unified View of Data", ACM Transactions on Database Systems, Vol. 1, No. 1, March 1976, pp. 9-36.

[Cher84] G. W. Cherry, Parallel Programming in ANSI Standard Ada. Reston, VA: Reston Publishing Co., 1984.

[Cher86] G. W. Cherry, The PAMELA Designer's Handbook. Thought**Tools, Reston, VA, Dec. 1986.

[Clem84] P. C. Clements, R. A. Parker, D. L. Parnas, J. E. Shore, and K. H. Britton, "A Standard Organization for Specifying Abstract Interfaces", Naval Research Laboratory, Washington, DC, June 14, 1984.

[Cohe86] P. F. Cohen, "The Transformation of Functional Decompositions to Object-Oriented Designs", presentation at Ada Expo '86, Charleston, WV, 1986.

[CoxB86] B. J. Cox, Object-Oriented Programming: An Evolutionary Approach. Don Mills, Ontario: Addison-Wesley Publishing Co., 1986.

[CSCo84] CSC, "Digital System Development Methodology (DSDM)", Version 2.0, Computer Sciences Corporation, March 1984.

INFORMATION ON SPECIFIC METHODS

[Dale86] P. C. Daley, "C3I Rapid Prototype Investigation", Rome Air Development Center, Griffiss Air Force Base, NY, Jan. 86.

[Davi83] C. G. Davis, S. Jajodia, P. A. Ng, and R. T. Yeh, Entity-Relationship Approach to Software Engineering. New York, NY: North Holland, 1983.

[Davi88a] A. Davis, "A Comparison of Techniques for the Specification of External System Behavior," Communications of the ACM, Vol. 31, No. 9, Sept. 1988, pp 1098-1115.

[deBo83] P. de Bondeli, "Models for the Control of Concurrency in Ada Based on Pr-T Nets", Proceedings of the Adatec-Ada Europe Joint Conference on Ada, edited by the Commission of the European Communities, Brussels, March 1983.

[deBo86] P. de Bondeli, "On the Use of Semantic Specification for the Verification and Validation of Real Time Software", Proceedings of the 3rd IDA Workship on Ada Verification, Research Triangle Institute, Research Triangle Park, NC, May 1986.

[DeMa78] T. DeMarco, Structured Analysis and System Specification. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1978.

[Deut88] M. Deutsch, "Focusing Real-Time Systems Analysis on User Operations," IEEE Software, Sept. 1988, pp. 39-50.

[Dick77] M. E. Dickover, C. L. McGowan, and D. T. Ross, "Software Design Using SADT", Proceedings of the ACM, Oct. 1977, pp. 125-133.

[Dixo88] D. Dixon, "Integrated Support for Project Management", Proceedings of the 10th International Conference on Software Engineering, Singapore, April 11-15, 1988, pp. 49-58.

[Dows87] M. Dowson, "Integrated Project Support with IStar", IEEE Software, Vol. 4, No. 6, Nov. 1987, pp. 6-15.

[DRIC00] Defence Research Information Centre, "Handbook of Mascot 3.1", Glasgow, Scotland.

[Dubo88] E. Dubois, J. Hagelstein, and A. Rifaut, "Formal Requirements with ERAE", Philips Journal of Research, V. 43, 3/4, 1988, pp. 393-414.

[EVBS85] EVB Software Engineering, Inc., "An Object Oriented Design Handbook for Ada Software", EVB, Jan. 1985.

[Faga76] M. E. Fagan, "Design and Code Inspections to Reduce Errors in Program Development", IBM Systems Journal, Vol. 15, No. 3, 1976, pp. 182-211.

[Fire86] D. G. Firesmith, "Object-Oriented Development", Magnavox Electronic Systems Co., Ft. Wayne, IN, Jan. 1986.

[Fran85] E. N. Frankowski, "A Summary of the RaPIER Project at One Year" (draft), Honeywell, Inc., Golden Valley, MI, Nov. 21, 1985.

[Gane79] C. Gane and T. Sarson, Structured Systems Analysis: Tools and Techniques. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1979.

INFORMATION ON SPECIFIC METHODS

[Garl88] D. Garlan, C. Krueger, and B. Staudt, "TransformGen: Automating the Maintenance of Structure-Oriented Environments", Carnegie Mellon Technical Report No. CMU-CS-88-186, 1988.

[Genr81] H. J. Genrich and K. Lautenbach, "System Modelling with High Level Petri Nets", Theoretical Computer Science, Vol. 13, 1981, pp. 109-136.

[Gilb85] T. Gilb, "Evolutionary Delivery vs the Waterfall Model", SIGSOFT Software Engineering Notes, Vol. 10, July 1985, pp. 49-61.

[Gilb87] T. Gilb and L. Krzanik, "Design by Objectives (DBO): A·Basis for Automated Software Engineering Design", ACM SIGSOFT Software Engineering Notes, Vol. 12, No. 2, April 1987, pp. 42-49.

[Gilb88] T. Gilb, Principles of Software Engineering Management. Addison-Wesley, 1988.

[Goma84] H. Gomaa, "A Software Design Method for Real Time Systems", Communications of the ACM, Sept. 1984.

[Goma86] H. Gomaa, "Software Development of Real-Time Systems", Communications of the ACM, Vol. 29, July 1986, pp. 657-668.

[Goma87] H. Gomaa, "Using the DARTS Software Design Method for Real Time Systems", Proceedings of the Twelfth Structured Methods Conference, Chicago, Aug. 1987.

[Goma88a] H. Gomaa, "Extending the DARTS Software Design Method to Distributed Real Time Applications", Proceedings of the 21st Hawaii International Conference on System Sciences, Jan. 1988.

[Goma88b] H. Gomaa, "Adarts - An Ada based Design Approach for Real Time Systems", Software Productivity Consortium Technical Report SPC-TR-88-021, Aug. 1988.

[Good78] D. I. Good, R. M. Cohen, C. G. Hoch, L. Hunter, and D. F. Hare, "1978 Report on the Language Gypsy, Version 2.0", Technical Report ICSCA-CMP-10, Certifiable Minicomputer, Project ICSCA, The University of Texas at Austin.

[Good85] D. I. Good, "Mechanical Proofs about Computer Programs", in C. A. R. Hoare and J. C. Shepardson, eds., Mathematical Logic and Programming Languages. Englewood Cliffs, NJ: Prentice-Hall, 1985, pp. 55-75.

[Grif84] W. G. Griffin, "Software Engineering in GTE", IEEE Computer, Vol. 17, Nov. 1984.

[Habe86] A.N. Habermann and D. Notkin, "Gandalf: Software Development Environments", IEEE Transactions on Software Engineering, Vol. SE-12, No. 12, Dec. 1986, pp. 1117-1127.

[Hagl88] J. Hagelstein, "Declarative Approach to Information Systems Requirements", Knowledge Based Systems, V. 1,4, 1988, pp. 211-220.

[Hage87] M. Hagemann, "Formal Requirements Specification of Process Control Systems," SIGSOFT Software Engineering Notes, Vol 12, No. 4, Oct. 1987, pp 36-42.

INFORMATION ON SPECIFIC METHODS

[Hage88] M. Hagemann, "Requirements Analysis for Realtime Automation Projects," Proceedings of the 10th International Conference on Software Engineering, The Computer Society Press, April, 1988, pp. 122-129.

[Hare87a] D. Harel, "Statecharts: A Visual Formalism for Complex Systems", Science of Computer Programming, 8.3, June 1987, pp. 231-274.

[Hare87b] D. Harel, A. Pnueli, J. P. Schmidt, and R. Sherman, "On the Formal Semantics of Statecharts", Proceedings of the 2nd IEEE Symposium on Logic in Computer Science, Ithaca, NY, June 22-24, 1987, published by IEEE Press, NY, 1987, pp. 54-64.

[Hare88] D. Harel et al, "STATEMATE: A Working Environment for the Development of Complex Reactive Systems", Proceedings of the Tenth IEEE Conference on Software Engineering, Singapore, April 13-15, 1988, published by IEEE Press, NY, 1988, pp. 396-406.

[Hatl85] D. J. Hatley, "A Structured Analysis Method for Real-Time Systems", presented at the Fall DECUS U. S. Symposium, Dec. 1985.

[Hatl87] D. J. Hatley and I. A. Pirbhai, Strategies for Real-Time System Specification. New York, NY: Dorset House Publishing Co., 1987.

[Haye85] I. Hayes, "Applying Formal Specifications to Software Development in Industry", IEEE Transactions on Software Engineering, Vol. SE-11, Feb. 1985.

[Heit88] M. Heitz, "Using Hierarchical Object Oriented Design for the Development of Distributed Ada Systems", Proceedings of Munich Ada-Europe 1988 Conference "Ada in Industry", Cambridge University Press, Ada Companion Series.

[Hest81] S. D. Hester, D. L. Parnas, and D. F. Utter, "Using Documentation as a Software Design Medium", Bell System Technical Journal, Vol 60, No. 8, Oct. 1981.

[Hevn88] A. R. Hevner, "An Integrated Systems Development Environment with Box Structures", paper presented at INTEC Symposium on Systems Analysis and Design, Atlanta, Oct./Nov. 1988.

[Higg87] D. A. Higgins, "Specifying Real-Time/Embedded Systems Using Feedback/Control Models", Proceedings of the Twelfth Structured Methods Conference, Chicago, IL, Aug. 3-6, 1987, pp. 124-147.

[Hoff88] D. Hoffman, and R. Snodgrass, "Trace Specifications: Methodology and Models," IEEE Transactions on Software Engineering, Vol. 14, No. 9, Sept. 1988, pp. 1243-1252.

[Hori72] S. Hori, CAM-I Long Range Planning Final Report for 1972. Chicago: Illinois Institute of Technology Research Institute, 1972.

[Horn73] J. J. Horning and B. Randell,"Process Structuring", Computing Surveys, Vol. 5, March 1973, pp. 5-30.

[Hrus86] P. Hruschka, PROMOD - Motivation and Introduction. Laguna Hills, California: Promod, Inc., 1986.

INFORMATION ON SPECIFIC METHODS

[IBMC74] IBM, "HIPO--A Design Aid and Documentation Technique", White Plains, NY, 1974.

[Iiva87] J. Iivari, "A Hierarchical Spiral Model for the Software Process: Notes on Boehm's Spiral Model", SIGSOFT Software Engineering Notes, Vol. 12, Jan. 1987, pp. 35-37.

[ISSI87] Integrated System Solutions, Inc., "Structured-JAD", internal publication, 1987.

[Jack75] M. Jackson, Principles of Program Design. New York, NY: Academic Press, 1975.

[Jaco87] I. Jacobson, "Object Oriented Development in an Industrial Environment", Proceedings of OOPSLA 1987 Conference.

[Jaco86] I. Jacobson, "Language Support for Changeable Large Real Time Systems", OOPSLA86, ACM, special issue of SIGPLAN Notices, Vol. 21, No. 11, Nov. 1986.

[Jawo86] W. Jaworski, I. MacCuaig, T. Marinelli, and T. Nyisztor "Executable Specification for a Large Industrial Process", Proceedings of the Canadian Conference on Industrial Computer Systems, (Montreal 1986), pp. 60-1 through 60-5, Canadian Industrial Computer Society, 1986.

[Jone86] C. B. Jones, Systematic Software Development using VDM. Prentice-Hall, 1986.

[Kaeh86] T. Kaehler and D. Patterson, "A Small Taste of Smalltalk", Byte, Aug. 1986.

[Kais88] G. Kaiser, P. Feiler, and S. Popovich, "Intelligent Assistance for Software Development and Maintenance," IEEE Software, May 1988, pp. 40-49.

[Kang83] H. Kangassalo, "CONCEPT D - A Graphical Formalism for Representing Concept Structures", in H. Kangassalo, ed., Second Scandinavian Research Seminar on Information Modelling and Data Base Management. Acta Universitatis Tamperensis, Ser. B, Vol. 19, Univ. of Tampere, Finland, 1983.

[Kang84] H. Kangassalo and P. Aalto, "Experiences on User Participation in the Development of a Conceptual Schema by Using a Concept Structure Interface", in B. Shackel, ed., INTERACT '84. Proceedings of the first IFIP Conference on Human Computer Interaction. North-Holland, Amsterdam, 1984.

[Kang88] H. Kangassalo, "CONCEPT D - A Graphical Language for Conceptual Modelling and Data Base Use", Proceedings of the 1988 IEEE Workshop on Visual Languages, University of Pittsburgh, Oct. 10-12, 1988.

[Kapl86] J. S. Kaplan, "Using Ada Design Language to Achieve Internal Reuse", presentation at Ada Expo '86, Charleston, WV, 1986.

[King85] M. J. King and J. P. Pardoe, Program Design Using JSP - A Practical Introduction. New York, NY: John Wiley and Sons, 1985.

[Knig81] C. Knight, L. Robertson, and L. Pink, "A Practical Implementation of Data Structured Systems Analysis and Design", Reader Enquiry Serivce, Jackson Method User Group, Amsterdam, Sept. 1981.

INFORMATION ON SPECIFIC METHODS

[Krie88a] B. Krieg-Bruckner, "The PROSPECTRA Methodology of Program Development, in Proceedings of IFIP/IFAC Working Conference on Hardware and Software for Real Time Process Control", Warsaw, 1988, North Holland publisher, pp. 257-271.

[Krie88b] B. Krieg-Bruckner, "Algebraic Formalisation of Program development by Transformation", in Proceedings of European Symposium on Programming, Nancy, March 1988, Springer-Verlag Lecture Notes in Computer Science No. 300, pp. 34-48.

[Krue88] E.W. Krueger and A. Bogliolo, "Scaling Up: Segmentation and Concurrency in Large Software Databases", Carnegie Mellon Technical Report No. CMU-Cs-87-178, Feb. 1988.

[Lefk85] D. Lefkowitz, "Design Methodology for Embedded Systems" (final report to Naval Air Development Center), Analytics, Inc., Willow Grove, PA, Nov. 1985.

[Luca87] P. Lucas, "VDM: Origins, Hopes, and Achievements", Proceedings, VDM-Europe Symposium '87, Lecture Notes in Computer Science, Springer Verlag, 1987.

[Luck84] D. Luckham and F. W. von Henke, "An Overview of Anna: A Specification Language for Ada", Stanford Univ., Stanford, CA, Sept. 1984.

[Luck86] D. C. Luckham, D. P. Helmbold, D. Bryan, and M. A. Haberler, "Task Sequencing Language for Specifying Distributed Ada Systems" (preliminary draft), Stanford Univ., Stanford, CA, Oct. 1986.

[Luck87] D. Luckman, R. Neff, and D. Rosenblum, "An Environment for Ada Software Development Based on Formal Specifications", SIGAda Ada Letters, Vol. 7, No. 3, May-June 1987.

[Lund81] M. Lundeberg, G. Godkuhl, and A. Nilsson, Information Systems Development - A Systematic Approach. Englewood Cliffs, N.J: Prentice-Hall, Inc., 1981.

[Luqi88a] Luqi, and V. Berzins, "Rapidly Prototyping Real-Time Systems," IEEE Software, Sept. 1988, pp. 25-36.

[Luqi88b] Luqi, V. Berzins, and R. Yeh, "A Prototyping Language for Real-Time Software," IEEE Transactions on Software Engineering, Vol. 14, No. 10, Oct. 1988, pp. 1409-1423.

[Lund81] M. Lundeberg, G. Godkuhl, and A. Nilsson, Information Systems Development - A Systematic Approach. Englewood Cliffs, N.J: Prentice-Hall, Inc., 1981.

[MacL86] B. MacLennan, Principles of Programming Languages, 2nd Ed. New York, NY: Holt, Reinhart and Winston, 198C.

[Mart88] C. F. Martin, User-Centered Requirements Analysis. Englewood Cliffs, NJ: Prentice-Hall, 1988.

[Mart88a] C. F. Martin, "Hierarchical Planning for Large Systems", System Builder Magazine, Dec. 1988/Jan. 1989, pp. 46-53.

[Mart88b] C. F. Martin, "Hierarchical Planning for Evolution of Large Information Systems", proceedings of CASE'88 Workshop, Cambridge, MA, July, 1988.

INFORMATION ON SPECIFIC METHODS

[McMe84] S. McMenamin and J. Palmer, Essential Systems Analysis. New York: Yourdon Press, 1984.

[McNe86] A. T. McNeile, "Jackson System Development", in Information Systems Design Methodologies: Improving the Practice, T. W. Olle, H. G. Sol and A. A. Verrijn-Stuart, eds., Elsevier Science Publishers B. V. (North-Holland), IFIP, 1986.

[Mend86] G. Mendal, "Designing for Ada Reuse: A Case Study," Computer Systems Laboratories, June 1986.

[Meye87] B. Meyer, "Reusability: The Case for Object-Oriented Design", IEEE Software, Vol. 4, March 1987.

[Mill87] H. Mills, M. Dyer, and R. Linger, "Cleanroom Software Development," IEEE Software, Sept. 1987, pp. 19-25.

[Mill88] H. Mills, "Stepwise Refinement and Verification in Box-Structured Systems," IEEE Computer, Vol. 21, No. 6, June 1988, pp. 23-36.

[Mull82] G. Mullery, "CORE, A Method for Controlled Requirements Specification", Proceedings of the 4th International Conference on Software Engineering, sponsored by the IEEE, 1982.

[Musa87] J. D. Musa, "Measuring and Managing Software Reliability", presentation notes, AT & T Bell Labs, Whippany, NJ, 1987.

[Myer74] G. Myers, "Structured Design", IBM Systems Journal, Vol. 13, No. 1, 1974.

[Myer75] G. J. Myers, Reliable Software through Composite Design. New York: Petrocelli/Charter, 1975.

[Myer83] G. J. Myers, Composite/Structured Design. New York, NY: Van Nostrand Reinhold Co., 1983.

[Nijs79] G. M. Nijssen, F. Van Assche, and J. Snijders, "End-User Tools for Information Systems Requirements Definition", in Formal Models and Practical Tools for Information System Design. New York, NY: North Holland Publishing Company, 1979.

[Notk85] D. Notkin, "The GANDALF Project", The Journal of Systems and Software, Vol. 5, No. 2, May 1985, pp. 91-105.

[ONei80] D. O'Neill, "Software Engineering Program", IBM Systems Journal, Vol. 19, No. 4, Dec. 1980.

[ONei83] D. O'Neill, "An Integration Engineering Perspective", The Journal of Systems and Software, Vol. 3, 1983, pp. 77-83.

[ONei86] D. O'Neill, "Software Engineering and Ada in Design", presented at the Washington Ada Symposium, March, 1986.

[OrrK77] K. Orr, Structured Systems Development. New York: Yourdon Press, 1977.

INFORMATION ON SPECIFIC METHODS

[OrrK83] K. Orr, Structured Requirements Definition. Topeka, KS: Ken Orr and Associates, 1983.

[Oste81] L. J. Osterweil, "Software Engineering Research: Directions for the Next Five Years", Computer, April 1981, pp. 36-37.

[Palm87] J. F. Palmer, "Integrating the Structured Techniques with JAD: Leveled Systems Development", a working paper presented at the Twelfth Structured Methods Conference, Aug. 1987, pp. 290-299.

[Palm88] J. F. Palmer, "CASE and ESA", Proceedings of the 14th Semiannual Seminar of the NY Chapter of the IEEE/CS, November 1988.

[Palm89] J. F. Palmer, Essential systems Development: A Fourth Generation Methodology, Seminar available at Technology Transfer Institute, Inc., Feb. 1989.

[Parn86] D. L. Parnas and P. C. Clements, "Rational Design Process: How and Why to Fake It," IEEE Transactions on Software Engineering, Vol. SE-12, Feb. 1986.

[Parn72] D. Parnas, "On the Criteria to be Used in Decomposing Systems into Modules", Communications of the ACM, Vol. 15, Dec. 1972.

[Parn78] D. Parnas, "Designing Software for Ease of Extension and Contraction", Proceedings of the 3rd International Conference on Software Engineering, The Computer Society Press, May 1978, pp. 184-195.

[Parn85] D. L. Parnas, P. C. Clements, and D. M. Weiss, "The Modular Structure of Complex Systems," IEEE Transactions on Software Engineering, Vol. SE-11, No. 3, March 1985.

[Part83] H. Partsch and R. Steinbruggen, "Program Transformation Systems", Computing Surveys, Vol. 15, No. 3, Sept. 1983, pp. 199-236. Reprinted in [Agre86].

[Pasc86] G. A. Pascoe, "Elements of Object-Oriented Programming", Byte, Aug. 1986.

[Pete87] L. Peters, Advanced Structured Analysis and Design. Englewood Cliffs, NJ: Prentice Hall, Inc., 1987.

[Petr81] J. L. Peterson, Petri Net Theory and the Modeling of Systems. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1981.

[Pete77] J. L. Peterson, "Petri nets", Computing Surveys, Vol. 9, No. 3, Sept 1977.

[Proi88] C. Proix, C. Cauvet, and C. Rolland, "An Expert System Approach for Information System Design", EDBT '88, Venice, 1988.

[Rajl85a] V. Rajlich, "Stepwise Refinement Revisited", Journal of Systems and Software, March 1985, pp. 80-88.

[Rajl85b] V. Rajlich, "Paradigms for Design and Implementation in Ada", Communications of the ACM, July 1985, pp. 718-727.

INFORMATION ON SPECIFIC METHODS

[Rajl87] V. Rajlich, "Refinement Methodology for Ada", IEEE Transactions on Software Engineering, Vol. SE-13, No. 4, April 1987, pp. 472-478.

[Reis86] "Petri Nets for Software Engineering", in Petri Nets: Applications and Relations to Other Models of Concurrency. Berlin: Springer-Verlag, 1986, pp. 63-96.

[Roch83] A. Rochfeld and H. Tardieu, "Merise: An Information System Design and Development Methodology", Information and Management 6, 1983, pp. 143-149.

[Roch86] A. Rochfeld, "An Information System Design and Development Methodology", Proceedings of the 5th International Conference on Entity Relationship Approach, Nov. 1986, to be published by North Holland.

[Roll82] C. Rolland and C. Richard, "REMORA", IFIP Conference on Comparative Review of Information Systems Design Methodologies (CRIS), 1982, North-Holland, publishers.

[Roll86] C. Rolland and C. Proix, "An Expert System Approach to Information System Design", IFIP Congress, Dublin, 1986, North-Holland, publishers.

[Roll88] C. Rolland et al., "The RUBIS System" in T. W. Olle, C. Bhabuta, and A. Verrign-Stuart, eds., CRIS 88 (Computerized Assistence During the Information Systems Life Cycle), North Holland, publishers.

[Ross76] D. T. Ross and J. W. Brackett, "An Approach to Structured Analysis", Computer Decisions, Vol. 8, No. 9, 1976, pp. 40-44.

[Ross77] D. Ross and K. Schoman, "Structured Analysis for Requirements Definitions", IEEE Transactions on Engineering, Vol. SE-3, Jan. 1977, pp. 6-15.

[Ross85a] D. T. Ross "Applications and Extensions of SADT", IEEE Computer, Vol. 18, No. 4, April 1985, pp. 25-34.

[Ross85b] D. T. Ross, "Douglas Ross Talks About Structured Analysis", IEEE Computer, Vol. 18, No. 7, July 1985, pp. 80-88.

[Rush85] G. Rush, "A Fast Way to Define System Requirements", ComputerWorld, October 7, 1985, pp.ID/11-16.

[Saya85] H. Sayani, "PSL/PSA: New Generation Real-Time Extensions", Oct. 1985 (presented at National Conference and Workshop on Methodologies and Tools for Real-Time Systems, Washington, D.C.,Oct. 28-Nov. 1, 1985).

[Scan87] J. M. Scandura, "A Cognitive Approach to Software Development: the PRODOC Environment and Associated Methodology", Journal of Pascal, Ada, and Modula-2, Vol. 6, No. 4, 1987.

[Schw75] J. Schwartz, "On Programming: An Interim Report of the SETL Project", Technical Report, Courant Institute, New York University, New York, 1975.

INFORMATION ON SPECIFIC METHODS

[Seid86] E. Seidewitz and M. Stark, "General Object-Oriented Software Development", Goddard Space Flight Center Software Engineering Lab Document SEL-86-002, Greenbelt, MD, Aug. 1986.

[Seid87a] E. Seidewitz and M. Stark, "Towards a General Object-Oriented Ada Life Cycle", Proceedings of the Joint Ada Technology Conference/Washington Ada Symposium, March 1987.

[Seid87b] E. Seidewitz and M. Stark, "Towards a General Object-Oriented Software Development Methodology", SIGAda Ada Letters, Vol. 7, No. 4, July-Aug. 1987, pp. 54-67.

[Seid88] E. Seidewitz, "General Object-Oriented Software Development; Background and Experience," Proceedings of the Hawaii International Conference on Systems Science, Jan. 1988.

[SEJ 86] Software Engineering Journal, May 1986: special edition on Mascot3 containing four papers covering design representation, the method, use with Ada, and testing. Jointly published by IEE/BCS in London.

[Shem87] I. Shemer, "Systems Analysis: A Systemic Analysis of a Conceptual Model", Communications of the ACM, Vol. 30, No. 6, June 1987.

[Shla88a] S. Shlaer and S. Mellor, Object-Oriented Systems Analysis: Modeling the World in Data. Englewood Cliffs, NJ: Prentice-Hall, 1988.

[Shla88b] S. Shlaer, S. Mellor, D. Ohlsen, and W. Hywari, "The Object-Oriented Method for Analysis", Proceedings of the Tenth Structured Development Forum (SDF-X), San Francisco, CA, August 1988.

[Siev85] G. E. Sievert and T. A. Mizell, "Specification-Based Software Engineering with TAGS", IEEE Computer, Vol. 18, No. 4, April 1985 pp. 56-65.

[Simo88a] R. Simonian and M. Crone, "InnovAda: An Object-Oriented Ada Enhancement", Presentations from the US Army ISEC Technology Strategies 1988 Conference, Alexandria, VA, Feb., 1988.

[Simo88b] R. Simonian and M. Crone, "InnovAda: True Object-Oriented Programming in Ada", The Journal of Object-Oriented Programming, (scheduled for winter 1988 publication).

[Smit88] M. K. Smith and S. R. Tockey, "An Integrated Approach to Software Requirements Definition Using Objects", Proceedings of the Tenth Structured Development Forum, San Francisco, Ca., August 8-11, 1988; also listed in Proceedings of Ada Expo '88, Anaheim, Ca., Oct. 9-12, 1988.

[Stev85] W. P. Stevens, "Using Data Flow for Application Development", Byte, June 1985.

[Stev82] W. P. Stevens, "How Data Flow Can Improve Application Development Productivity", IBM Systems Journal, Vol. 21, No. 2, 1982, pp. 162-178. (Reprint Order No.G321-5165)

[Stev81] W. P. Stevens, Using Structured Design. New York: John Wiley and Sons, 1981.

INFORMATION ON SPECIFIC METHODS

[Stev74] W. P. Stevens, G. J. Myers, and L. L. Constantine, "Structured Design", IBM Systems Journal, Vol. 13, No. 2, May 1974, pp. 115-139.

[Tate85] G. Tate and T. W. G. Docker, "A Rapid Prototyping System Based on Data Flow Principles", SIGSOFT Software Engineering Notes, Vol. 10, April 1985, pp. 28-34.

[Teic77] D. Teichroew and E. A. Hershey, III, "PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems", IEEE Transactions on Software Engineering, Vol. 3, No. 1, Jan. 1977, pp. 41-48.

[Tele87] Teledyne Brown Engineering, "Technology for the Automated Generation of Systems (TAGS): Overview", Teledyne Brown Engineering, Huntsville, AL, 1987.

[Terw86] R. B. Terwilliger and R. H. Campbell, "Please: Executable Specifications for Incremental Software Development", University of Illinois, Oct. 1986.

[Theu86] N. Theuretzbacher, "HCDM: A Hierarchical Design Method for CHILL based Systems", Proceedings of 1986 International Zurich Seminar on Digital Communications, IEEE Catalog No. 86CH2277-2, pp.163-169.

[Thom88] D. A. Thomas and K. Johnson, "Orwell - A Configuration Management System for Team Programming", Proceedings of ACM OOPSLA '88, San Diego, CA, 1988, pp. 135-141.

[TRWC77] TRW, "The Software Engineering Library -- DCDS Documentation", Jan. 1977.

[TRW 87a] TRW System Development Division, Distributed Comupting Design System: "A Technical Overview". Huntsville, AL, 25 July 1987.

[TRW 87b] TRW System Development Division, Distributed Comupting Design System: "Tools User's Guide with Appendices". Huntsville, AL, Oct. 1987.

[TRW 87c] TRW System Development Division, Distributed Comupting Design System: "Methodology Guide with Appendices". Huntsville, AL, Oct. 1987.

[Tsai88] J. P. Tsai and J. C. Ridge, "Intelligent Support for Specification Transformations", IEEE Software, Vol. 5, No. 6, Nov. 1988, pp. 28-35.

[VanA83] F. Van Assche, D. Simons, and M. Vanhoedenaghe, "The Automated Mapping from a Binary Conceptual Schema to a 5th NF Data Base Schema", published by the International Center for Information Analysis Services, Control Data Belgium Inc., 1983.

[VanG82] J. J. Van Griethuysen, editor, "Concepts and Terminology for the Conceptual Schema and the Information Base", Report of ISO TC/97/SC5/WG3, March 1982.

[Verh82] G. M. A. Verhegen and J. Van Bekkum, "NIAM: An Information Analysis Method", Proceedings of the IFIP/TC8 Working Conference on a Comparative Review of Information Systems Design Methodologies, Noordwykerhout, Nethederlands, May 1982, North Holland Publishing Co.

[Warn74] J. Warnier, Logical Construction of Programs. New York: Van Nostrand Reinhold, 1974.

INFORMATION ON SPECIFIC METHODS

[Wall87] R. H. Wallace, J. E. Stockenberg, and R. N. Charette, A Unified Methodology for Developing Systems. Intertext Publications, New York: McGraw-Hill, 1987.

[Ward85] P. T. Ward and S. J. Mellor, Structured Development for Real-Time Systems. New York: Yourdon Press, 1985.

[Ward86] P. T. Ward, "The Transformation Schema: An Extension of the Data Flow Diagram to Represent Control and Timing", IEEE Transactions on Software Engineering, Vol. SE-12, No. 2, Feb. 1986.

[Ward89] P. T. Ward, "How to Integrate Object-Oriented Design with Structured Analysis and Design", IEEE Software, Vol. 6, No. 2, March 1989, pp. 74-82.

[Warn81] J. Warnier, Logical Construction of Systems. New York: Van Nostrand Reinhold, 1981.

[Wass86] A. I. Wasserman, P. A. Pircher, D. T. Shewmake, and M. L. Kerster, "Developing Interactive Information Systems with User Software Engineering Methodology", IEEE Transactions on Software Engineering, Vol. SE-12, No. 2, Feb. 1986, pp. 326-345.

[Wass87] A. I. Wasserman and P. A. Pircher, "A Graphical, Extensible Integrated Environment for Software Development", ACM SIGPLAN Notices, Vol. 22, No. 1, 1987, pp. 131-142.

[Wass88] A. I. Wasserman, P. A. Pircher, et. al., "A Generalized Annotation Mechanism for Objects in a CASE Environment", Proceedings of Software Engineering and its Applications, Toulouse, France, Dec. 1988.

[Wass89] A. I. Wasserman, P. A. Pircher, and R. J. Muller, "An Object-Oriented Structured Design Method for Code Generation", SIGSOFT Software Engineering Notes, Vol. 14, No. 1, Jan. 1989, pp. 32-55.

[West87] S. J. Westfold, L. Z. Markosian, and W. A. Brew, "Knowledge-Based Software Development from Requirements to Code", unpublished report supplied by Reasoning Systems Inc. Palo Alto, CA.

[Wirt71] N. Wirth, "Program Development by Stepwise Refinement", Communications of the ACM, Vol. 14, No. 4, April 1971, pp. 221-227.

[Your79] E. Yourdon and L. Constantine, Structured Design. Englewood Cliffs, NJ: Prentice-Hall, 1979.

[Your86a] E. Yourdon, Managing the Structured Techniques Strategies for Software Development in the 1990's. New York: Yourdon Press, 1986.

[Your86b] E. Yourdon, "What Ever Happened to Structured Analysis?", Datamation, June 1, 1986.

[Your89] E. Yourdon, Modern Structured Analysis. Englewood Cliffs, NJ: Yourdon Press, 1989.

[Zave82] P. Zave, "An Operational Approach to Requirements Specification for Embedded Systems", IEEE Transactions on Software Engineering, Vol. 8, No. 3, May 1982, pp. 250-269.

INFORMATION ON SPECIFIC METHODS

[Zave84] P. Zave, "The Operational Versus the Conventional Approach to Software Development", Communications of the ACM, Vol. 27, No. 2, Feb. 1984, pp. 104-118.

[Zave86] P. Zave and W. Schell, "Salient Features of an Executable Specification Language and Its Environment", IEEE Transactions on Software Engineering, Vol. 12, No. 2, Feb. 1986, pp. 312-325.

APPENDIX B

SURVEY
of
SOFTWARE METHOD DEVELOPERS

Software Method Name:
(long form)

Software Method Acronym:
(If no standard acronym,
enter suggested abbreviated
form for use in catalog)

Principal Architects/Developer(s)
of the method:

Provider of Method
(Individual and/or company):

Address/phone number:

If respondent is not developer/provider:

Respondent's Name:

Respondent's Title:

Address/phone number:

Length and type of
experience with method:

NOTE: Survey answers should reflect the method as it is currently being used for the development of deliverable systems.

1. For each of the following ACTIVITIES associated with software system development, describe the level of support provided by this method by circling ONE of the following:

S	The method prescribes SPECIFIC directions and procedures for conducting the activity.
G	The method provides GUIDELINES or a framework for the activity (e.g., a document template).
R	The method REQUIRES the activity but does not provide directions for accomplishing the activity.
N	The method does NOT ADDRESS the activity.

a. Requirements definition/clarification	S	G	R	N
b. System specification	S	G	R	N
c. System design	S	G	R	N
d. System implementation/installation	S	G	R	N
e. Software quality assurance	S	G	R	N
f. Risk/cost assessment	S	G	R	N
g. Other project management	S	G	R	N
h. Other ______________________________	S	G	R	N

2. Many methods conform to an underlying approach or strategy that suggests a general way of beginning or conducting the development effort. For each of the following, circle the ONE code that BEST reflects the relationship of the method to the approach.

F	The method is FOUNDED upon the approach.
C	The method is COMPATIBLE with the approach.
I	The method is INCOMPATIBLE with the approach.
U	The relationship of the method to the approach is UNKNOWN.

a. Data flow-oriented approach (Structured Analysis/Design)	F	C	I	U
b. Data structure-oriented approach	F	C	I	U
c. Object-oriented approach	F	C	I	U
d. Control-oriented approach	F	C	I	U
e. Event-oriented approach	F	C	I	U
f. Entity-relationship modeling	F	C	I	U
g. Binary-relationship modeling	F	C	I	U
h. Functional decomposition	F	C	I	U
i. Other ______________________________	F	C	I	U

3. To what degree does the method fit into the context of the following software process paradigms? Circle ONE code in each row:

W The method is WELL-SUITED.

C The method is COMPATIBLE.

I The method is INAPPROPRIATE.

N NO OPINION.

a. Waterfall (classic sequential software process model with feedback loops)	W	C	I	N
b. Boehm's spiral (alternative evaluation & risk analysis at increasing levels of elaboration)	W	C	I	N
c. Rapid prototyping (executable model of system subset)	W	C	I	N
d. Incremental model (includes evolutionary development)	W	C	I	N
e. Operational model (behavior simulation)	W	C	I	N
f. Transformational model (program construction by application of transformation rules)	W	C	I	N
g. 4GL (4th generation languages with automatic code generation)	W	C	I	N
h. Other: ______________________________	W	C	I	N

4. Is the most effective use of this method dependent upon ONE particular software process paradigm? ___Yes ___No

If yes, which one? ____________________________________

5. In what year was the method first used for the development of a deliverable system?

____

6. Is the method available for general use? ___Yes ___No

If no, what restrictions apply? ________________________________

7. Circle the estimated number of delivered systems that have been developed using this method:

Less than 5 5-20 21-100 101-250 More than 250

8. Circle the estimated number of organizations that have used this method:

Less than 5 5-20 21-50 51-100 More than 100

9. Indicate the SUITABILITY of the method with respect to each application area below by circling ONE of the following:

W The method is WELL-SUITED for use in this area.

C The method is COMPATIBLE for use in this area.

I The method is INAPPROPRIATE for use in this area.

N NO OPINION on method's suitability.

a. Embedded systems/process control	W	C	I	N
b. Time critical/real-time	W	C	I	N
c. Scientific/engineering	W	C	I	N
d. Systems programming (operating system software, etc.)	W	C	I	N
e. Distributed processing/networks	W	C	I	N
f. Data processing/database	W	C	I	N
g. Expert systems/artificial intelligence	W	C	I	N
h. Image processing/pattern recognition	W	C	I	N
i. Large scale simulation/modeling	W	C	I	N
j. Other ______________________________	W	C	I	N

10. Give some specific types of applications for which delivered systems have been developed using this method:

11. Describe the relationship of the method to project size by circling ANY of the following which apply:

S SMALL projects (e.g., controller for a bank of elevators).

M MEDIUM projects (e.g., communication network manager).

L LARGE projects (e.g., Space Station system).

a. For what size projects is the method intended?	S	M	L
b. For what size projects has the method been used?	S	M	L

12. What implementation language(s) have been most frequently used when coding systems developed with this method?

13. Indicate the relationship of the method to the concepts or programming practices listed below by circling ONE of the following:

E The concept/practice is ESSENTIAL to the method.

C The concept/practice is COMPATIBLE with the method.

I The concept/practice is INCONSISTENT with the method.

U The relationship of the concept/practice is UNKNOWN.

a. Stepwise refinement	E	C	I	U
b. Information hiding	E	C	I	U
c. Process abstraction	E	C	I	U
d. Abstract data-types	E	C	I	U
e. Structured programming	E	C	I	U
f. Genericity	E	C	I	U
g. Inheritance	E	C	I	U
h. Use of assertions (pre- and post-conditions)	E	C	I	U
i. Module coupling/cohesion	E	C	I	U

14. Use the following code to indicate the extent to which the method utilizes the techniques listed below:

R	The technique is REQUIRED/essential to the method.
E	The technique is strongly ENCOURAGED by the method.
N	The technique is NOT ADDRESSED by the method.

a. To what extent is each of the following techniques utilized to clarify system requirements or behavior? (Circle ONE for each technique.)

1) Rapid prototyping	R	E	N
2) Dynamic animation	R	E	N
3) Simulation	R	E	N
4) Incremental/Evolutionary development	R	E	N
5) Executable specifications	R	E	N
6) Other ______________________	R	E	N

b. To what extent is each of the following analysis and review techniques utilized by this method? (Circle ONE for each technique.)

1) Data-structure analysis	R	E	N
2) Data-flow analysis	R	E	N
3) Control-flow analysis	R	E	N
4) Decision tables	R	E	N
5) Formal proof techniques	R	E	N
6) Design reviews	R	E	N
7) Code walk-throughs	R	E	N
8) Facilitated Application Specification Techniques	R	E	N
9) Change Control Board review	R	E	N
10) Other ______________________	R	E	N

15. Use the following code to indicate the extent to which the method uses the modes of representation listed below:

R	The mode is REQUIRED by the method.
E	The mode is strongly ENCOURAGED by the method.
C	The mode is COMPATIBLE with the method.
I	The mode is INCONSISTENT with the method.
U	The relationship of the mode to the method is UNKNOWN.

a. To what extent is each of the following TEXTUAL modes of representation utilized by this method? (Circle ONE for each mode.)

						Check when method provides automated support
1) Specified documentation templates	R	E	C	I	U	___
2) Narrative overviews of modules	R	E	C	I	U	___
3) Structured English	R	E	C	I	U	___
4) Program design language	R	E	C	I	U	___
5) Formal specification languages	R	E	C	I	U	___
6) Mathematical notation	R	E	C	I	U	___
7) Warnier/Orr diagrams	R	E	C	I	U	___
8) Decision tables	R	E	C	I	U	___
9) Other ____________________	R	E	C	I	U	___

b. To what extent is each of the following ICONOGRAPHICAL modes of representation utilized by this method? (Circle ONE for each mode.)

1) Finite-state diagrams	R	E	C	I	U	___
2) Petri nets	R	E	C	I	U	___
3) Data-flow diagrams	R	E	C	I	U	___
4) Control-flow diagrams	R	E	C	I	U	___
5) Entity-relationship diagrams	R	E	C	I	U	___
6) Flowcharts	R	E	C	I	U	___
7) HIPO charts	R	E	C	I	U	___
8) Hierarchy charts	R	E	C	I	U	___
9) Leighton diagrams	R	E	C	I	U	___
10) Nassi-Shneiderman charts	R	E	C	I	U	___
11) Buhr diagrams	R	E	C	I	U	___
12) Booch diagrams	R	E	C	I	U	___
13) PG structure graphs	R	E	C	I	U	___
14) Other ____________________	R	E	C	I	U	___

16. If the method uses several modes of expression, does the method prescribe mapping rules for translating from one mode to another?

___Yes ___No

If yes, identify those pairs for which a mapping rule is prescribed (e.g., data-flow diagrams ===> module hierarchy):

17. In your opinion, what specific features or aspects of this method are designed to facilitate and coordinate communication within the development team?

18. In your opinion, what specific features or aspects of this method are designed to facilitate and coordinate communication between management and the technical development team?

19. In your opinion, what specific features or aspects of this method are designed to facilitate and coordinate communication between the client and the development organization?

20. If the following documents are required to be produced when using this method, indicate 1) the type of format provided and 2) the automated support, if provided, using the following codes.

Format (circle appropriate format if document is required):

F A FIXED document format is prescribed by the method.

T The document format is TAILORABLE within the method.

N The method requires the document, but does NOT PRESCRIBE the format.

Automated support (circle appropriate support if provided):

G Document is automatically GENERATED based on data produced from other step(s) in the method.

P Document is produced from user responses to computer-directed PROMPTS.

WHEN APPROPRIATE, circle the code(s) for each document type below:

		Format		Automated Support	
a. Requirements definition	F	T	N	G	P
b. Functional specification	F	T	N	G	P
c. Behavioral specification	F	T	N	G	P
d. Architectural specification	F	T	N	G	P
e. Interface specification	F	T	N	G	P
f. System structure chart	F	T	N	G	P
g. Data dictionary	F	T	N	G	P
h. Design document	F	T	N	G	P
i. Internal program documentation	F	T	N	G	P
j. Quality assurance/Test plan	F	T	N	G	P
k. Support/Installation plan	F	T	N	G	P
l. User manual	F	T	N	G	P
m. Design decision log	F	T	N	G	P
n. Problem log	F	T	N	G	P
o. Change log	F	T	N	G	P
p. Other ______________________	F	T	N	G	P

21. For each of the following test activities, describe the level of support the method provides for assessing conformance of the developing software to the required system. Circle ONE code in each row and indicate if automated support is available from the method vendor.

S The method prescribes SPECIFIC directions and procedures for conducting the activity.

G The method provides GUIDELINES or a framework for the activity (e.g., a document template).

R The method REQUIRES the activity but does not provide directions for accomplishing the activity.

N The method does NOT ADDRESS the activity.

					Check when method provides automated support
a. Test planning at one or more precise points in the software process	S	G	R	N	___
b. Generation of tests based on system requirements	S	G	R	N	___
c. Unit/integration testing	S	G	R	N	___
d. Field testing/Acceptance testing	S	G	R	N	___
e. Generation of test data	S	G	R	N	___
f. Regression testing	S	G	R	N	___
g. Prescriptive checking of interfaces	S	G	R	N	___
h. Other ______________________________	S	G	R	N	___

22. For each of the following, circle the ONE code which BEST reflects the level of support provided by the method relative to maintaining a traceable record of technical decision-making during the software development process:

A The method provides AUTOMATED recording procedures.

S The method provides SPECIFIC directions for recording information.

R The method REQUIRES that a record be kept, but does not provide specific directions for recording information.

N The method does NOT specifically require that a record be maintained.

a. A record is maintained of specification/design options which were considered.	A	S	R	N
b. A record is maintained of any trade-off studies.	A	S	R	N
c. A record is maintained of the rationale for any decision.	A	S	R	N
d. A record is maintained of the personnel who were involved in making a decision.	A	S	R	N
e. A record is maintained of all changes related to specification/design decisions.	A	S	R	N

23. Does the method prescribe steps for handling the following types of requirements of the target system? If yes, provide a brief description of how the method addresses this.

a. Timing constraints. ___Yes ___No.

b. Spatial constraints. ___Yes ___No.

c. Special features of the target hardware architecture. ___Yes ___No.

d. Special features of the target operating system. ___Yes ___No.

e. Concurrency issues. ___Yes ___No.

f. Fault-tolerance issues. ___Yes ___No.

g. Security of access. ___Yes ___No.

24. Use the following scale to indicate the level of support provided by the method for the project management activities listed below:

S The method prescribes SPECIFIC directions and procedures for conducting the activity.

G The method provides GUIDELINES or a framework for the activity.

R The method REQUIRES the activity but does not provide directions for accomplishing the activity.

N The method does NOT ADDRESS the activity.

a. For each of the following ACTIVITIES involving risk assessment and cost estimation of software development, describe the level of support provided by this method. (Circle ONE for each activity.)

					Check when method provides automated support
1) Analyzing risk	S	G	R	N	___
2) Assessing complexity	S	G	R	N	___
3) Estimating initial cost	S	G	R	N	___
4) Projecting cost of completion	S	G	R	N	___
5) Providing incremental data about expenditures	S	G	R	N	___
6) Tracking project progress	S	G	R	N	___
7) Other ______________________	S	G	R	N	___

b. For each of the following other ACTIVITIES associated with project management, describe the level of support provided by this method. (Circle ONE for each activity.)

1) Project planning	S	G	R	N	___
2) Scheduling and/or manpower loading	S	G	R	N	___
3) Generation of Pert or Gantt charts	S	G	R	N	___
4) Allocation of personnel to tasks	S	G	R	N	___
5) Allocation of development resources	S	G	R	N	___
6) Configuration management	S	G	R	N	___
7) Reliability estimation	S	G	R	N	___
8) Other ______________________	S	G	R	N	___

25. What assistance is available to train an organization in the use of the method? Check ALL that apply:

___ a. Hands-on demonstrations

___ b. Overview presentations

___ c. Classroom tutorials

___ d. On-site consulting by the vendor

___ e. On-site consulting by independent consultants

___ f. On-line tutorials

___ g. Video tapes

___ h. On-line help facility

___ i. "Hot line" service

___ j. User manuals

___ k. Users' support group

___ l. Related publications from third-parties

___ m. Periodic technical updates (e.g., newsletters, bulletins)

26. In your opinion, how many DAYS would be required for a project manager to acquire an understanding of the major features and benefits of the method? ____________

27. In your opinion, how many DAYS would be required for an experienced developer (5 or more years' practice) to learn to use the essentials of the method? ____________

28. In your opinion, how many MONTHS would be required for an experienced developer to achieve the level of expert user of the method? ____________

29. When applicable, estimate the following unit costs (in dollars):

	single user	low volume user	high volume user	does not apply
a. Unit cost to acquire method and required components				
b. Unit cost for other recommended components supplied by method vendor				
c. Unit cost for technical training				
d. Unit cost for management overview				

30. If this method is supported by automated tools, what hardware/software configuration is appropriate for hosting them?

31. Is there a licensing policy
(e.g., site license, workstation license)? ___Yes ___No.

If yes, briefly elaborate:

32. List below any additional automated tools, which tools specifically support activities of this method and would be of interest to method users:

Name of Tool	Tool Vendor	Activities Supported

33. In your opinion, what are the minimum qualifications needed by a development team leader in order to successfully use the method? For each of the following, check ONE.

a. Minimum college-level technical education?

___Less than 2 years

___2-3 years

___Bachelor's degree

___Advanced degree.

b. Minimum number of years of development experience?

___0

___1-2

___3-5

___6-10

___More than 10.

c. Number of programming languages for which the person has a working knowledge?

___1

___2

___3-4

___5 or more.

d. Number of different software systems with which the person has had development/programming/maintenance experience?

___1

___2

___3-4

___5 or more.

34. List the major theoretical construct(s) which, in your opinion, should be understood by an experienced developer in order to successfully use the method (e.g., finite-state machines, regular grammars, etc.).

When applicable, for questions 35 through 44, provide a brief, but specific, description of the relevant aspect(s) of the method. These answers will serve as a basis for the description of key characteristics of the method.

35. Specifically, how does the method assist in reducing the effort needed to fully incorporate changes in the requirements?

36. Specifically, how does the method assist in ensuring that consistency is maintained among specification, design or code when changes are made to any of these three entities?

37. Specifically, how does the method assist in the early detection of inconsistencies and/or errors?

38. Specifically, how does the method facilitate the transformation across phases of the software process? (e.g., from specification to design, or from design to code)

39. Specifically, what procedures does the method provide for involving the client during the software development process, and where does this involvement occur?

40. Specifically how does the method assist in porting end-product systems to different target configurations?

41. Specifically, how does the method assist in identifying possible reusable components such as design or code?

42. What steps in the "traditional" development process can be skipped as a result of using this method?

How is this accomplished?

43. Briefly describe the major aspect(s) of the method which differentiate this method from others directed toward the same application domain. Be specific.

44. If this method is based upon or an extension of pre-existing methods, please name these methods:

45. Please provide bibliographic references to comprehensive descriptions/discussions of the method. The referenced source material should be available to the public.

46. Please provide an overview of the method that describes the flow of events occurring as software is developed, adding any salient features and assumptions that have not been covered by this questionnaire. Also describe the function of any necessary automated tools that have not yet been mentioned. Your answer will serve as the primary source of information for the Overview Section of the method writeup. Feel free to provide this information in outline form. (You may substitute a copy of a previously published description which provides such an overview.)

(Optional) Coaments on this survey:

APPENDIX C

SURVEY
of
EMERGING METHODS

Software Method Name:____________________________________
(long form)

Software Method Acronym:__________________________________
(If no standard acronym,
enter suggested abbreviated
form for use in catalog)

Principal Architects/Developer(s)
of the method:

____

____

Respondent for method
(Individual and/or company):____________________________

Address/phone number:________________________________

____

____

NOTE: Survey answers should reflect the method as it is currently planned for use in the immediate future.

1. For each of the following ACTIVITIES associated with software system development, describe the level of support provided by this method by circling ONE of the following:

S The method prescribes SPECIFIC directions and procedures for conducting the activity.

G The method provides GUIDELINES or a framework for the activity (e.g., a document template).

R The method REQUIRES the activity but does not provide directions for accomplishing the activity.

N The method does NOT ADDRESS the activity.

a. Requirements definition/clarification	S	G	R	N
b. System specification	S	G	R	N
c. System design	S	G	R	N
d. System implementation/installation	S	G	R	N
e. Software quality assurance	S	G	R	N
f. Risk/cost assessment	S	G	R	N
g. Other project management	S	G	R	N
h. Other ______________________________	S	G	R	N

2. Many methods conform to an underlying approach or strategy that suggests a general way of beginning or conducting the development effort. For each of the following, circle the ONE code that BEST reflects the relationship of the method to the approach.

F The method is FOUNDED upon the approach.

C The method is COMPATIBLE with the approach.

I The method is INCOMPATIBLE with the approach.

U The relationship of the method to the approach is UNKNOWN.

a. Data flow-oriented approach (Structured Analysis/Design)	F	C	I	U
b. Data structure-oriented approach	F	C	I	U
c. Object-oriented approach	F	C	I	U
d. Control-oriented approach	F	C	I	U
e. Event-oriented approach	F	C	I	U
f. Entity-relationship modeling	F	C	I	U
g. Binary-relationship modeling	F	C	I	U
h. Functional decomposition	F	C	I	U
i. Other ______________________________	F	C	I	U

3. To what degree does the method fit into the context of the following software process paradigms? Circle ONE code in each row:

W	The method is WELL-SUITED.
C	The method is COMPATIBLE.
I	The method is INAPPROPRIATE.
N	NO OPINION.

a. Waterfall (classic sequential software process model with feedback loops)	W	C	I	N
b. Boehm's spiral (alternative evaluation & risk analysis at increasing levels of elaboration)	W	C	I	N
c. Rapid prototyping (executable model of system subset)	W	C	I	N
d. Incremental model (includes evolutionary development)	W	C	I	N
e. Operational model (behavior simulation)	W	C	I	N
f. Transformational model (program construction by application of transformation rules)	W	C	I	N
g. 4GL (4th generation languages with automatic code generation)	W	C	I	N
h. Other: ______________________________	W	C	I	N

4. Is the most effective use of this method dependent upon ONE particular software process paradigm? ___Yes ___No

If yes, which one? ____________________________________

5. Which of the following describes the current availability of the method? Check the ONE statement below which BEST applies:

___ Method is in development.

___ Method is in beta test.

___ Method is in exploratory research stage.

6. What year is the anticipated target for releasing the method for general use?

____

7. Indicate the SUITABILITY of the method with respect to each application area below by circling ONE of the following:

W	The method is WELL-SUITED for use in this area.
C	The method is COMPATIBLE for use in this area.
I	The method is INAPPROPRIATE for use in this area.
N	NO OPINION on method's suitability.

a. Embedded systems/process control	W	C	I	N
b. Time critical/real-time	W	C	I	N
c. Scientific/engineering	W	C	I	N
d. Systems programming (operating system software, etc.)	W	C	I	N
e. Distributed processing/networks	W	C	I	N
f. Data processing/database	W	C	I	N
g. Expert systems/artificial intelligence	W	C	I	N
h. Image processing/pattern recognition	W	C	I	N
i. Large scale simulation/modeling	W	C	I	N
j. Other ________________________________	W	C	I	N

8. Is this method intended to address a specific type of application area/problem?

___Yes ___No

If yes, please indicate which area/problem:

9. What implementation language(s), if any, are tied to the method?

10. Indicate the relationship of the method to the concepts or programming practices listed below by circling ONE of the following:

E The concept/practice is ESSENTIAL to the method.

C The concept/practice is COMPATIBLE with the method.

I The concept/practice is INCONSISTENT with the method.

U The relationship of the concept/practice is UNKNOWN.

a. Stepwise refinement	E	C	I	U
b. Information hiding	E	C	I	U
c. Process abstraction	E	C	I	U
d. Abstract data-types	E	C	I	U
e. Structured programming	E	C	I	U
f. Genericity	E	C	I	U
g. Inheritance	E	C	I	U
h. Use of assertions (pre- and post-conditions)	E	C	I	U
i. Module coupling/cohesion	E	C	I	U

11. List the major theoretical construct(s) which, in your opinion, should be understood by an experienced developer in order to successfully use the method (e.g., finite-state machines, regular grammars, etc.).

12. Use the following code to indicate the extent to which the method utilizes the techniques listed below:

R The technique is REQUIRED/essential to the method.

E The technique is strongly ENCOURAGED by the method.

N The technique is NOT ADDRESSED by the method.

a. To what extent is each of the following techniques utilized to clarify system requirements or behavior? (Circle ONE for each technique.)

1) Rapid prototyping	R	E	N
2) Dynamic animation	R	E	N
3) Simulation	R	E	N
4) Incremental/Evolutionary development	R	E	N
5) Executable specifications	R	E	N
6) Other ______________________________	R	E	N

b. To what extent is each of the following analysis and review techniques utilized by this method? (Circle ONE for each technique.)

1) Data-structure analysis	R	E	N
2) Data-flow analysis	R	E	N
3) Control-flow analysis	R	E	N
4) Decision tables	R	E	N
5) Formal proof techniques	R	E	N
6) Design reviews	R	E	N
7) Code walk-throughs	R	E	N
8) Facilitated Application Specification Techniques	R	E	N
9) Change Control Board review	R	E	N
10) Other ______________________________	R	E	N

13. Use the following code to indicate the extent to which the method uses the modes of representation listed below:

R	The mode is REQUIRED/essential to the method.
E	The mode is strongly ENCOURAGED by the method.
N	The mode is NOT USED by the method.

a. To what extent is each of the following TEXTUAL modes of representation utilized by this method? (Circle ONE for each mode.)

1) Specified documentation templates	R	E	N
2) Narrative overviews of modules	R	E	N
3) Structured English	R	E	N
4) Program design language	R	E	N
5) Formal specification languages	R	E	N
6) Mathematical notation	R	E	N
7) Warnier/Orr diagrams	R	E	N
8) Decision tables	R	E	N
9) Other ______________________	R	E	N

b. To what extent is each of the following ICONOGRAPHICAL modes of representation utilized by this method? (Circle ONE for each mode.)

1) Finite-state diagrams	R	E	N
2) Petri nets	R	E	N
3) Data-flow diagrams	R	E	N
4) Control-flow diagrams	R	E	N
5) Entity-relationship diagrams	R	E	N
6) Flowcharts	R	E	N
7) HIPO charts	R	E	N
8) Hierarchy charts	R	E	N
9) Leighton diagrams	R	E	N
10) Nassi-Shneiderman charts	R	E	N
11) Buhr diagrams	R	E	N
12) Booch diagrams	R	E	N
13) PG structure graphs	R	E	N
14) Other ______________________	R	E	N

14. If the method uses several modes of expression, does the method prescribe mapping rules for translating from one mode to another? ___Yes ___No

If yes, identify those pairs for which a mapping rule is prescribed (e.g., data-flow diagrams ===> module hierarchy):

15. For each of the following test activities, describe the level of support the method provides for assessing conformance of the developing software to the required system. Circle ONE code in each row and indicate if automated support is available from the method vendor.

S The method prescribes SPECIFIC directions and procedures for conducting the activity.

G The method provides GUIDELINES or a framework for the activity (e.g., a document template).

R The method REQUIRES the activity but does not provide directions for accomplishing the activity.

N The method does NOT ADDRESS the activity.

					Check when method provides automated support
a. Test planning at one or more precise points in the software process	S	G	R	N	___
b. Generation of tests based on system requirements	S	G	R	N	___
c. Unit/integration testing	S	G	R	N	___
d. Field testing/Acceptance testing	S	G	R	N	___
e. Generation of test data	S	G	R	N	___
f. Regression testing	S	G	R	N	___
g. Prescriptive checking of interfaces	S	G	R	N	___
h. Other ______________________	S	G	R	N	___

16. Does the method prescribe steps for handling specific types of requirements of the target system (e.g., timing or spatial constraints, concurrency or fault-tolerance issues, security of access)? If so, provide a brief description of how the method addresses any such requirements.

17. What project management activities are supported by the method by means of prescriptive directives and/or automated tools?

18. Briefly describe the hardware/software configuration(s) needed to support any automated facilities provided by the method.

19. If the following documents are required to be produced when using this method, indicate 1) the type of format provided and 2) the automated support, if provided, using the following codes.

Format (circle appropriate format if document is required):

F A FIXED document format is prescribed by the method.

T The document format is TAILORABLE within the method.

N The method requires the document, but does NOT PRESCRIBE the format.

Automated support (circle appropriate support if provided):

G Document is automatically GENERATED based on data produced from other step(s) in the method.

P Document is produced from user responses to computer-directed PROMPTS.

WHEN APPROPRIATE, circle the code(s) for each document type below:

	Format			Automated Support	
a. Requirements definition	F	T	N	G	P
b. Functional specification	F	T	N	G	P
c. Behavioral specification	F	T	N	G	P
d. Architectural specification	F	T	N	G	P
e. Interface specification	F	T	N	G	P
f. System structure chart	F	T	N	G	P
g. Data dictionary	F	T	N	G	P
h. Design document	F	T	N	G	P
i. Internal program documentation	F	T	N	G	P
j. Quality assurance/Test plan	F	T	N	G	P
k. Support/Installation plan	F	T	N	G	P
l. User manual	F	T	N	G	P
m. Design decision log	F	T	N	G	P
n. Problem log	F	T	N	G	P
o. Change log	F	T	N	G	P
p. Other ____________________	F	T	N	G	P

20. In your opinion, what are the minimum qualifications needed by a development team leader in order to successfully use the method? For each of the following, check ONE.

a. Minimum college-level technical education?

___Less than 2 years

___2-3 years

___Bachelor's degree

___Advanced degree.

b. Minimum number of years of development experience?

___0

___1-2

___3-5

___6-10

___More than 10.

c. Number of programming languages for which the person has a working knowledge?

___1

___2

___3-4

___5 or more.

d. Number of different software systems with which the person has had development/programming/maintenance experience?

___1

___2

___3-4

___5 or more.

When applicable, for questions 21 through 27, provide a brief, but specific, description of the relevant aspect(s) of the method. These answers will serve as a basis for the description of key characteristics of the method.

21. Specifically, how does the method assist in reducing the effort needed to fully incorporate changes in the requirements?

22. Specifically, how does the method assist in ensuring that consistency is maintained among specification, design or code when changes are made to any of these three entities?

23. Specifically, how does the method assist in the early detection of inconsistencies and/or errors?

24. Specifically, how does the method facilitate the transformation across phases of the software process? (e.g., from specification to design, or from design to code)

25. Specifically, what procedures does the method provide for involving the client during the software development process, and where does this involvement occur?

26. Specifically, how does the method assist in identifying possible reusable components such as design or code?

27. What steps in the "traditional" development process can be skipped as a result of using this method? Specify how this is accomplished.

28. If this method is based upon or an extension of pre-existing methods, please name these methods:

29. Please provide bibliographic references to comprehensive descriptions/discussions of the method. The referenced source material should be available to the public.

30. Please provide an overview of the method that describes the flow of events occurring as software is developed, adding any salient features and assumptions that have not been covered by this questionnaire. In particular, describe those specific aspects/features of the method that make this method different from currently existing methods, especially other methods directed toward the same application domain. Your answer will serve as the primary source of information for the Overview Section of the method writeup. Feel free to provide this information in outline form. (You may substitute a copy of a previously published description which provides such an overview.)

(Optional) Comments on this survey: