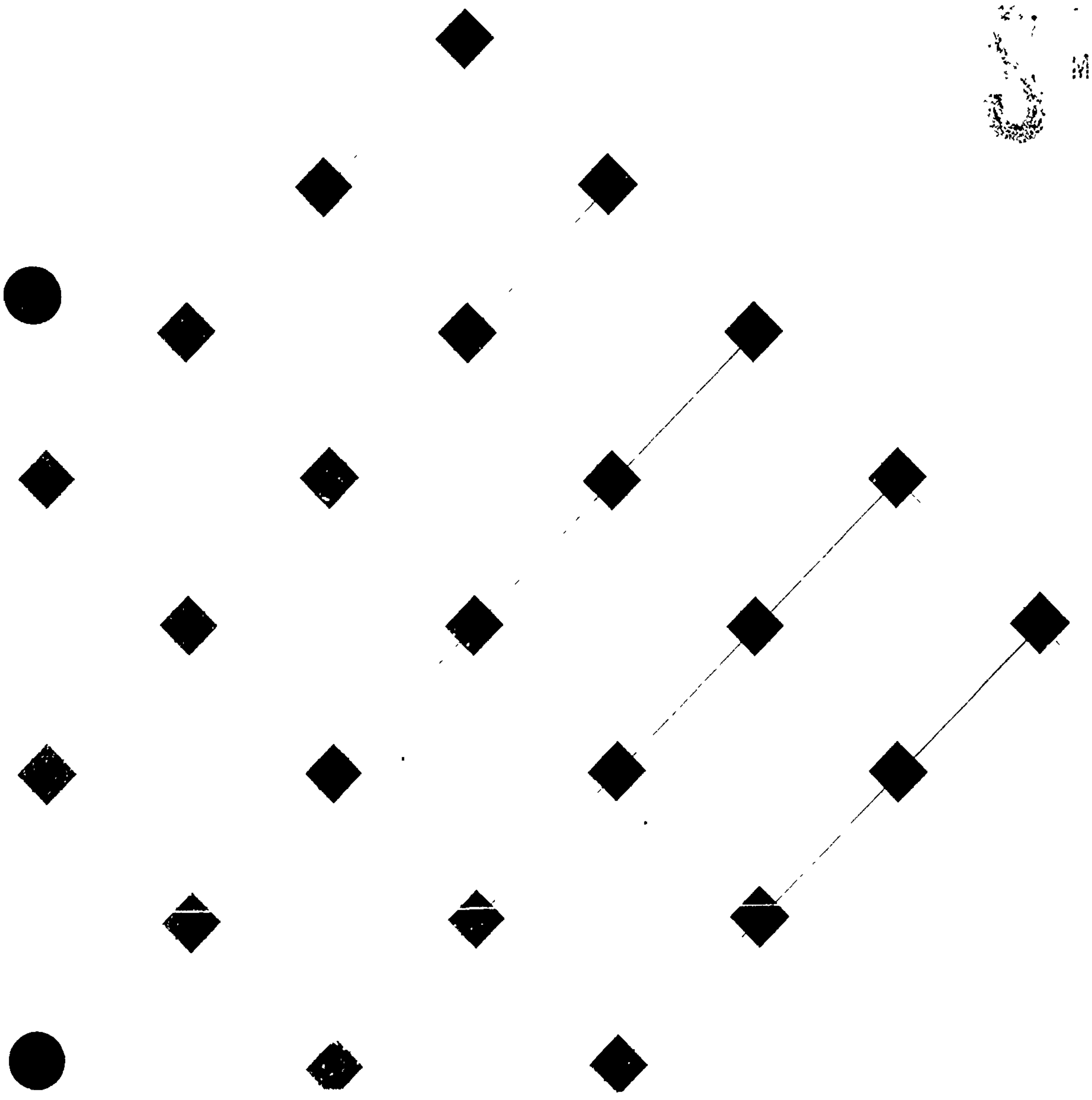
(2)

Carnegie Mellon University
Software Engineering Institute

Software Requirements

Curriculum Module SEI-CM-19-1.2



Software Requirements

SEI Curriculum Module SEI-CM-19-1.2

January 1990

John W. Brackett
Boston University

Carnegie Mellon University
Software Engineering Institute

This work was sponsored by the U.S. Department of Defense.
Approved for public release. Distribution unlimited.

This document was prepared for the

SEI Joint Program Office
ESD/AVS
Hanscom AFB, MA 01731

Review and Approval

This document has been reviewed and is approved for publication.

FOR THE COMMANDER

[illegible]

JOHN S. HERMAN, Capt, USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

Software Requirements

Acknowledgements

Dieter Rombach, of the University of Maryland, played an invaluable role in helping me organize this curriculum module. He reviewed many drafts. Lionel Deimel, of the SEI, was of great assistance in clarifying my initial drafts. Douglas T. Ross, one of the true pioneers in software engineering, taught me much of what I know about software requirements definition.

Contents

Capsule Description 1
Philosophy 1
Objectives 4
Prerequisite Knowledge 4
Module Content 5
  Outline 5
  Annotated Outline 5
Teaching Considerations 15
  Suggested Course Types 15
  Teaching Experience 15
  Suggested Reading Lists 15
Bibliography 17

Software Requirements

Module Revision History

Version 1.2 (January 1990)	Minor revisions and corrections
Version 1.1 (December 1989)	Revised and expanded bibliography; other minor changes Approved for publication
Version 1.0 (December 1988)	Draft for public review

Software Requirements

Capsule Description

This curriculum module is concerned with the definition of software requirements—the software engineering process of determining what is to be produced—and the products generated in that definition. The process involves all of the following:

- requirements identification
- requirements analysis
- requirements representation
- requirements communication
- development of acceptance criteria and procedures

The outcome of requirements definition is a precursor of software design.

Philosophy

The subject of software requirements is often given far less attention in software engineering education than software design, even though its importance is widely recognized. For example, Brooks [Brooks87] has written:

> The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems. No part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later.

The purpose of this module is to provide a comprehensive view of the field of software requirements in order that the subject area can be more widely taught. The module provides the material needed to understand the requirements definition process and the products produced by it. It emphasizes what must be done during requirements definition, independent of the specific techniques used. The material presented here should be considered prerequisite to the study of specific requirements methodologies and representation techniques.

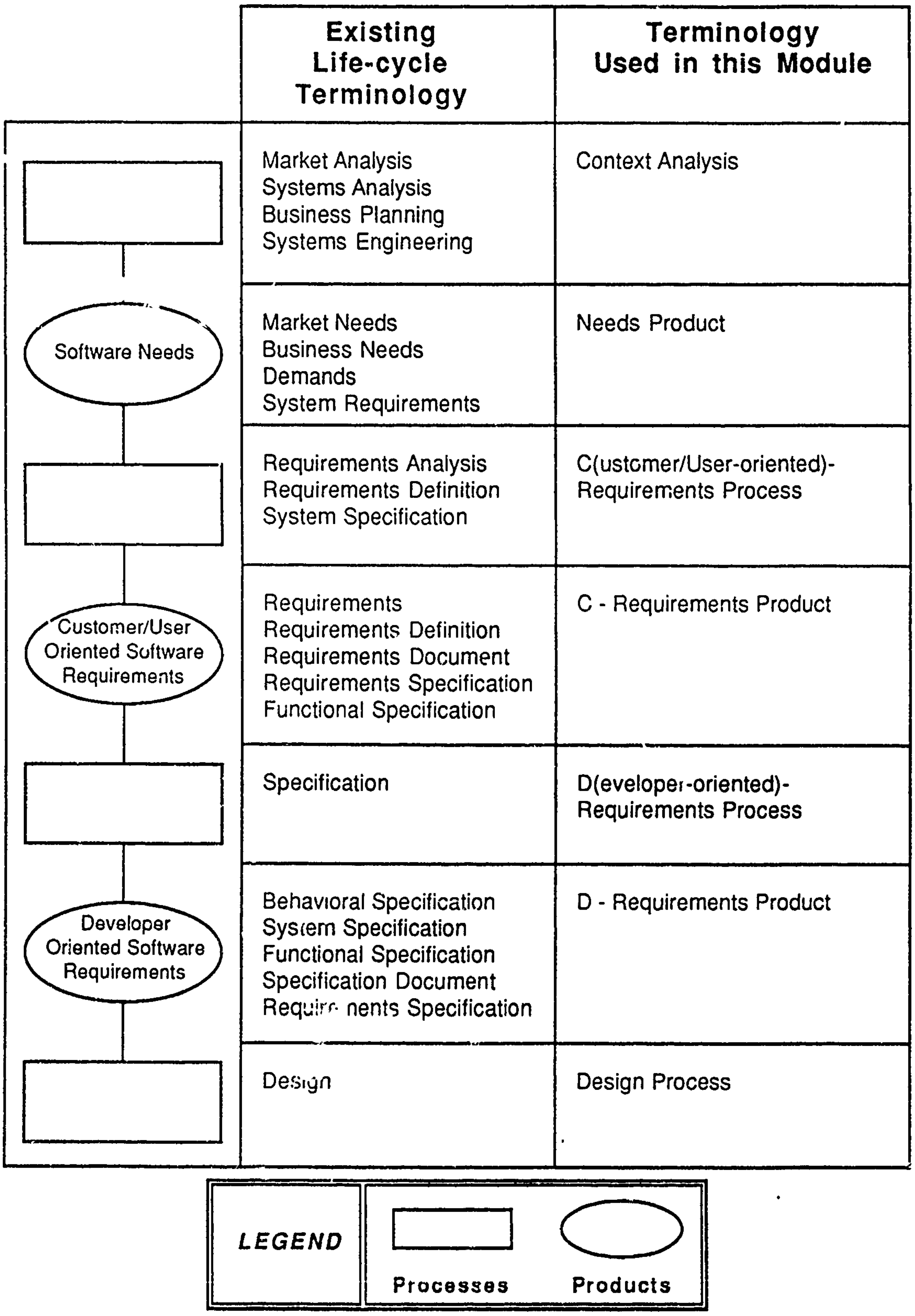
Software Requirements has been developed in conjunction with *Software Specification: A Framework* [Rombach90] and uses the conceptual framework and terminology presented in that module. This terminology is summarized in Figure 1. Both modules identify two products of the software requirements process: customer/user-oriented software requirements ("C-requirements") and developer-oriented software requirements ("D-requirements"). The principal objective of these documents is the achievement of agreement on what is to be produced. Their form, however, is largely determined by the communication needs of the diverse participants in the requirements process. The development of the D-requirements refines and augments the C-requirements, in order to provide the information required to support software design and subsequent validation of the developed software against the requirements.

Because of the dependence of this module on *Software Specification: A Framework*, that curriculum module should be read before studying this one.

This module reflects two strong opinions of the author:

- The software requirements definition process is highly dependent upon the previous steps in the system development process.
- The prime objective of the requirements definition process is to achieve agreement on what is to be produced.

Where the overall *system* requirements have been determined and the decision has been made that certain system functions are to be performed by software, the software requirements process is highly constrained by previous systems engineering work. In this situation, requirements are obtained largely


Software Needs
Customer/User Oriented Software Requirements
Developer Oriented Software Requirements
LEGEND
Processes
Products

Existing Life-cycle Terminology	Terminology Used in this Module
Market Analysis Systems Analysis Business Planning Systems Engineering	Context Analysis
Market Needs Business Needs Demands System Requirements	Needs Product
Requirements Analysis Requirements Definition System Specification	C(ustomer/User-oriented)- Requirements Process
Requirements Requirements Definition Requirements Document Requirements Specification Functional Specification	C - Requirements Product
Specification	D(eveloper-oriented)- Requirements Process
Behavioral Specification System Specification Functional Specification Specification Document Requirements Specification	D - Requirements Product
Design	Design Process

Figure 1. Life-cycle terminology used in this module.

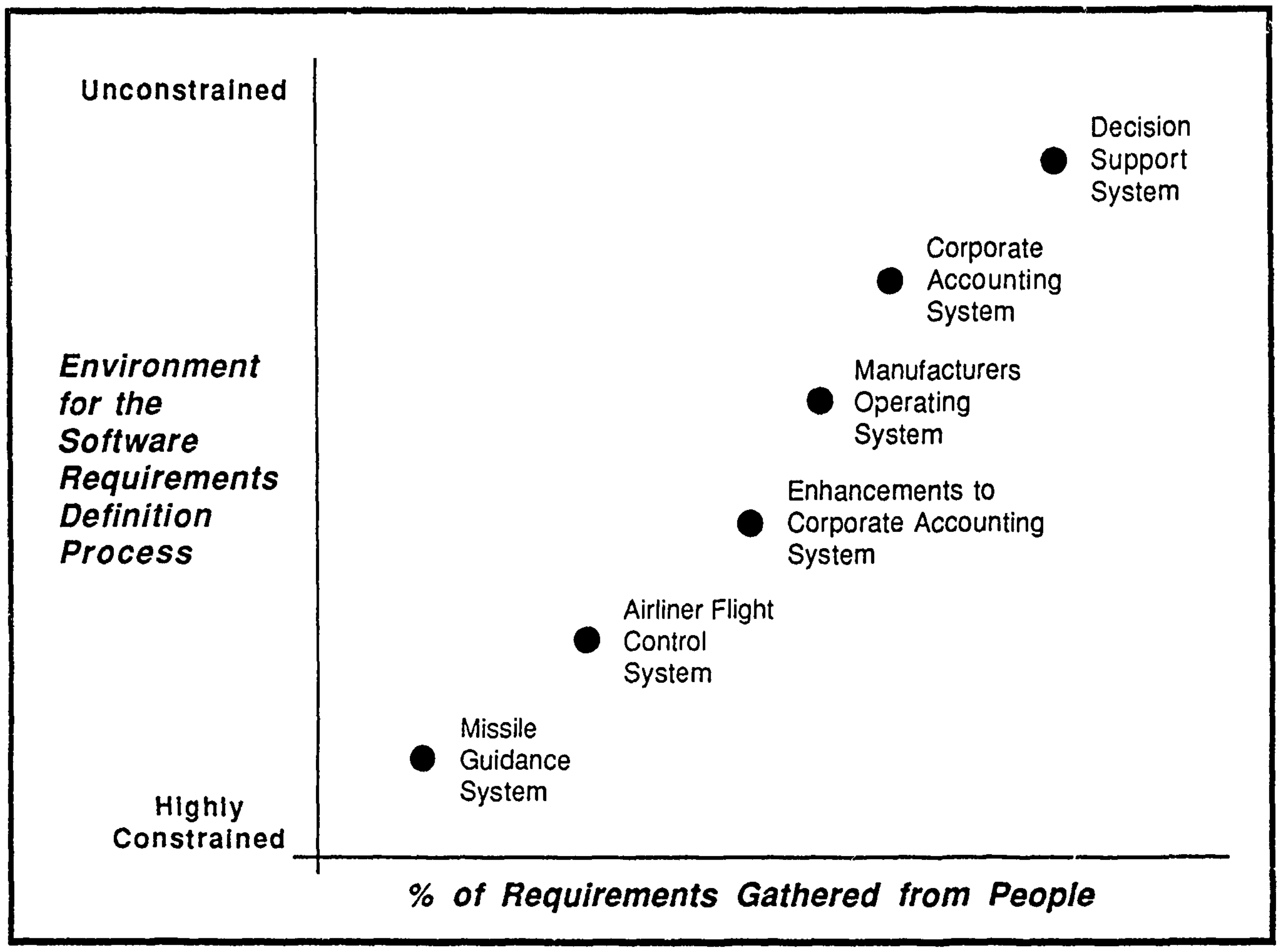


Figure 2. Sources of requirements.

by analyzing documents. (A typical example is the requirements definition for software to control a specific hardware device.)

Where there are few constraints imposed by the environment in which the software will operate and there may be many opinions on desired software functionality, requirements definition primarily involves eliciting requirements from people. (A typical example is the requirements definition needed to build decision support software for use by a group of .managers.)

Figure 2 suggests how the fraction of requirements elicited from people increases as constraints on the software requirements process decrease.

The fact that the prime objective of the requirements definition process is to achieve agreement on what is to be produced makes it mandatory that the products of the process serve to communicate effectively with the diverse participants.

Requirements products have three, sometimes competing, objectives:

1. To achieve agreement regarding the requirements between system developers, customers, and end-users.
2. To provide the basis for software design.
3. To support verification and validation.

During C-requirements development, the first objective is paramount. Later in the life cycle, the other two objectives increase in importance.

Objectives

The following is a list of possible objectives for instruction based upon this module. The objectives for any particular unit of instruction may include all of these or consist of some subset of this list, depending upon the nature of the unit and the backgrounds of the students.

Comprehension

- The student will be able to describe the products produced by requirements definition, the type of information each should contain, and the process used to produce the products.

Synthesis

- The student will be able to develop a plan for conducting a requirements definition project requiring a small team of analysts.
- The student will be able to perform requirements definition as part of a team working with a small group of end-users.

Evaluation

- The student will be able to evaluate critically the completeness and utility, for a particular audience, of requirements documents upon which software design is to be based.

Prerequisite Knowledge

Familiarity with the terms and concepts of the software engineering life cycle.

Module Content

Outline

I. Introduction to Software Requirements
   1. What are requirements?
   2. The requirements definition process
   3. Process participants and their roles
   4. The products of requirements definition

II. The Software Requirements Definition Process
   1. Requirements identification
      a. Software needs as input to requirements definition
      b. Elicitation from people
      c. Deriving softwa.: requirements from system requirements
      d. Task analysis to develop user interface requirements
   2. Identification of software development constraints
   3. Requirements analysis
      a. Assessment of potential problems
      b. Classification of requirements
      c. Evaluation of feasibility and risks
   4. Requirements representation
      a. Use of models
      b. Roles for prototyping
   5. Requirements communication
   6. Preparation for validation of software requirements
   7. Managing the requirements definition process

III. Software Requirements Products
   1. Results of requirements definition
      a. Functional requirements
      b. Non-functional requirements
      c. Inverse requirements (what the software *shall not* do)
      d. Design and implementation constraints
   2. Standards for requirements documents
   3. Customer/user-oriented software requirements
      a. Objectives
      b. Relative importance of specification attributes
      c. Key contents
   4. Developer-oriented software requirements
      a. Objectives
      b. Relative importance of specification attributes
      c. Key contents

IV. Techniques and Tools for Performing Software Requirements Definition
   1. Techniques for eliciting requirements from people
   2. Modeling techniques
   3. Representative requirements definition methods
      a. Structured Analysis and SADT
      b. DSSD
      c. SREM/DCDS
      d. NRL/SCR
   4. Computer support tools for model development
      a. Method-specific tools
      b. Non–method-specific tools
   5. Computer support tools for prototyping

Annotated Outline

I. Introduction to Software Requirements
   1. What are requirements?

There are many definitions of ***requirements***, which differ in their emphasis. The IEEE software engineering glossary [IEEE83] defines *requirement* as:

> (1) A condition or capability needed by a user to solve a problem or achieve an objective. (2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document. The set of all requirements forms the basis for subsequent development of the system or system component.

Abbott [Abbott86] defines *requirement* as:

> any function, constraint, or other property that must be provided, met, or satisfied to fill the needs of the system's intended user(s).

For projects in which the software development is highly constrained by prior system engineering work, the second IEEE definition is most applicable.

In less constrained environments, the first IEEE definition or Abbott's definition is appropriate.

Requirements cover not only the desired functionality of a system or software product, but also address non-functional issues (*e.g.*, performance, interface, and reliability requirements), constraints on the design (*e.g.*, operating in conjunction with existing hardware or software), and constraints on the implementation (*e.g.*, cost, must be programmed in Ada). The process of determining requirements for a system is referred to as *requirements definition*.

Our concern in this module is with *software requirements*, those requirements specifically related to software systems or to the software components of larger systems. Where software is a component of another system, software requirements can be distinguished from the *system requirements* of the larger artifact.

Software Specification: A Framework [Rombach90] describes several different software evolution (life-cycle) models and notes that the commonalities among these models are the types of products they eventually produce. The products shown in Figure 1 are assumed to be produced on a project, irrespective of how the products get built. Although the requirements on a small project may be defined informally and briefly, a definition of what the software development process will produce is required in all life-cycle models.

2. The requirements definition process

The requirements definition process comprises these steps:

- Requirements identification
- Identification of software development constraints
- Requirements analysis
- Requirements representation
- Requirements communication
- Preparation for validation of software requirements

These steps, which are not necessarily performed in a strictly sequential fashion, are described and discussed in Section II.

3. Process participants and their roles

[Rombach90] describes classes of project participants and their responsibilities. Rombach distinguishes between customers, who contract for the software project, and end-users, who will install, operate, use, and maintain the system incorporating the software. Customers are assumed to be responsible for acceptance of the software. Other participants he defines are: requirements analysts, who develop the C-requirements product; specification engineers, who evolve the C-requirements product into the D-requirements product; managers; and verification and validation (V&V) personnel.

Requirements analysts act as catalysts in identifying requirements from the information gathered from many sources, in structuring the information (perhaps by building models), and in communicating draft requirements to different audiences. Since there is a variety of participants involved in the requirements definition process, requirements must be presented in alternative, but consistent, forms that are understandable to different audiences.

4. The products of requirements definition

The outcome of requirements definition is the formulation of functional requirements, non-functional requirements, and design and implementation constraints. These requirements and constraints must be represented in a manner fulfilling the information needs of different audiences.

The following are the principal objectives of requirements products:

- **To achieve agreement on the requirements.** Requirements definition, the process culminating in the production of requirements products, is a communications-intensive one that involves the iterative elicitation and refinement of information from a variety of sources, usually including end-users with divergent perceptions of what is needed. Frequent review of the evolving requirements by persons with a variety of backgrounds is essential to the convergence of this iterative process. Requirements documents must facilitate communication with end-users and customers, as well as with software designers and personnel who will test the software when developed.
- **To provide a basis for software design.** Requirements documents must provide precise input to software developers who are not experts in the application domain. However, the precision required for this purpose is frequently at odds with the need for requirements documents to facilitate communication with other people.
- **To provide a reference point for software validation.** Requirements documents are used to perform software validation, *i.e.*, to determine if the developed software satisfies the requirements from which it was developed. Requirements must be stated in measurable form, so tests can be developed to show unambiguously whether each requirement has been satisfied [Boehm84a].

In formulating requirements, it is important for the analyst to maintain his role as analyst and avoid becoming a designer. Of two requirements purporting to represent the same need, the better one is that which allows the designer greater latitude. This advice is difficult to heed, both because users and customers often state particular design solutions as "needs" and because it is usually easier to postulate a solution in lieu of understanding what is really needed by users or customers. The analyst's objective should always be to maximize the options available to the designer. This objective can be well-illustrated by a simple example from another domain. The statement "the customer requires an automobile" provides the problem-solver with fewer options than "the customer needs a means to get to Cleveland."

II. The Software Requirements Definition Process

The steps in software requirements definition and the management of the process are discussed below.

1. Requirements identification

Requirements identification is the step of requirements definition during which software requirements are elicited from people or derived from system requirements [Davis82, Martin88, Powers84]. An important precursor to requirements definition is the context analysis process, which precedes requirements definition. (See Figure 1.)

a. Software needs as input to requirements definition

Context analysis [Ross77b] documents why software is to be created and why certain technical, operational, and economic feasibilities establish boundary conditions for the software development process. According to Ross, context analysis should answer the following questions:

- Why is the software to be created?
- What is the environment of the software to be created?
- What are the technical, operational, and economic boundary conditions that an acceptable software implementation must satisfy?

Context analysis for software to be developed for internal company use is frequently called *business planning* or *systems analysis*. In any case, we will refer to the product of context analysis as *software needs.*

Software needs can take significantly different forms depending upon the context of system development. In many situations, software needs will be very informal and will provide little detailed information for beginning requirements definition. For a highly constrained environment, *e.g.*, where software is embedded in a larger hardware system (an embedded system), system-level documentation frequently provides the context for software requirements definition. This documentation, which serves as software needs, typically covers system requirements, the allocation of system functions to software, and the description of interfaces between hardware and software.

When a software product is being developed for a heterogeneous audience (*e.g.*, a database manager or a spreadsheet package), software needs will typically contain the results of a market analysis and a list of important product features. Since information provided in software needs differs so widely, requirements definition must usually include an understanding of the environment in which the software will operate and how the software will interact with that environment.

b. Elicitation from people

An essential step in most requirements definition projects is elicitation of requirements-related information from end-users, subject-matter experts, and customers. *Elicitation* is the process performed by analysts for gathering and understanding information [Leite87]. Elicitation involves fact-finding, validating one's understanding of the information gathered, and communicating open issues for resolution.

Fact-finding uses mechanisms such as interviews, questionnaires, and observation of the operational environment of which the software will become a part.

Validation involves creating a representation of the elicitation results in a form that will focus attention on open issues and that can be reviewed with those who provided information. Possible representations include summary documents, usage scenarios, prototype software [Boehm84b], and models.

Some type of explicit approval to proceed with requirements definition completes the elicitation process. The audiences who must approve the requirements should agree that all relevant information sources have been contacted.

c. Deriving software requirements from system requirements

Requirements are created for embedded software based upon the system requirements for the system or system component in which the software is embedded. Traceability techniques are used to communicate how the software requirements relate to the system requirements, since the customer is usually more familiar with the system requirements. Because functions are allocated to software and hardware before software requirements definition begins, most of the functions the software is to perform will not be derived through requirements elicitation from end-users or customers.

d. Task analysis to develop user interface requirements

User Interface Development [Perlman88] describes methods for user interface evaluation that also apply to determining software requirements concerned with human interaction. Analyzing the tasks the user must perform should result in a detailed understanding of how a person is supposed to use the proposed software.

2. Identification of software development constraints

During this step, constraints on the software development process are identified. Typical constraints include cost, the characteristics of the hardware to which the software must interface, existing software with which the new software must operate, fault tolerance objectives, and portability requirements. Only software solutions satisfying the requirements and implemented within the restrictions imposed by the constraints are acceptable.

3. Requirements analysis

Requirements are generally gathered from diverse sources, and much analysis (*requirements analysis*) is usually needed before the results of requirements definition are adequate for the customer to commit to proceeding with further software development. "Adequate," in this context, means there is perceived to be an acceptable level of risk regarding technical and cost feasibility and an acceptable level of risk regarding the completeness, correctness, and lack of ambiguity in the results.

The principal steps in requirements analysis, which are frequently iterated until all issues are resolved, are:

a. Assessment of potential problems

This is the process step during which requirements are assessed for feasibility and for problems such as ambiguity, incompleteness, and inconsistency. Software requirements for embedded software must be verified to ensure they are consistent with the system requirements.

b. Classification of requirements

Requirements should be classified into priority categories such as mandatory, desirable, and inessential. "Mandatory" means that the software will not be acceptable to the customer unless these requirements are met in an agreed manner.

It is frequently useful also to assess requirements regarding stability; a stable requirement addresses a need that is not expected to change during the life of the software. Knowing that a requirement may change facilitates developing a software design that isolates the potential impact of the change.

c. Evaluation of feasibility and risks

Assessment of feasibility involves technical feasibility (i.e, can the requirements be met with current technology?), operational feasibility (i.e, can the software be used by the existing staff in its planned environment?), and economic feasibility (*i.e.*, are the costs of system implementation and use acceptable to the customer?) [Ross77b].

4. Requirements representation

Requirements representation is the step of requirements definition during which the results of requirements identification are portrayed. Requirements have traditionally been represented in a purely textual form. Increasingly, however, techniques such as model building and prototyping, which demand more precision in their description, are being used.

a. Use of models

Models are built during requirements definition to define specific characteristics of the software (*i.e.*, the functions it will perform and the interfaces to its environment) in a form that can be more easily understood and analyzed than a textual description. A good model:

- Reduces the amount of complexity that must be comprehended at one time.
- Is inexpensive to build and modify compared to the real thing.
- Facilitates the description of complex aspects of the real thing.

Most requirements methods include the development of models of some type to portray the results of requirements elicitation and to facilitate the requirements analysis process. An important motivation for building models during requirements definition is the belief that the model notation—and computer support tools supporting the notation—help the analyst identify potential problems early in the requirements definition process.

b. Roles for prototyping

Prototyping is frequently used to provide early feedback to customers and end-users and to improve communication of requirements between users and system developers. Many users find it difficult to visualize how software will perform in their environment if they have only a non-executable description of requirements. A prototype can be an effective mechanism to convey a sense of how the system will work. Hands-on use of a prototype is particularly valuable if a system has to be used by a wide variety of users, not all of whom have participated in the requirements definition process. Although a prototype is not a substitute for a thorough written specification, it allows representation of the effect of requirements—certain kinds of requirements, at any rate —with an immediacy not matched by its more static counterpart. Of course, not all elements of a system can be captured in a prototype at a reasonable cost.

In many situations, users do not understand the required functionality well enough to completely articulate their needs to analysts. A prototype based on the information obtained during requirements elicitation is often very useful in refining the required functionality [Gomaa81]. Models developed after requirements elicitation can be useful in deciding what functionality to include in such a prototype.

Boehm [Boehm86] describes possible roles of prototyping to minimize development risks due to incomplete requirements. Clapp [Clapp87] describes the uses of prototypes during requirements definition to assess technology risks and to assess whether a user interface can be developed that will allow the designated personnel to operate the system effectively. Modeling methods are of little help in determining the requirements for user interfaces. Development of prototypes of alternative user interfaces is usually required to obtain meaningful feedback from customers and users. If the technical feasibility of meeting essential requirements is in question, developing prototypes incorporating key algorithms can provide results that are not otherwise available.

5. Requirements communication

Requirements communication is the step in which results of requirements definition are presented to diverse audiences for review and approval. The fact that users and analysts are frequently expert in their own areas but inexperienced in each other's domains makes effective communication particularly difficult. The result of requirements communication is frequently a further iteration through the requirements definition process in order to achieve agreement on a precise statement of requirements.

6. Preparation for validation of software requirements

During this step, the criteria and techniques are established for ensuring that the software, when produced, meets the requirements. The customer and software developers must reach agreement on the proposed acceptance criteria and the techniques to be used during the software validation process, such as execution of a test plan to determine that the criteria have been met [Collofello88a].

7. Managing the requirements definition process

Requirements definition can present a major project management challenge. Nearly all cost and schedule estimating approaches assume that the requirements are defined and can be used to estimate roughly the size of the project. The effort for a requirements project is related to the total development man-months, and the effort rises in proportion to the number of divergent sources from which requirements information must be gathered and reconciled. For example, an application that must support five different classes of users with significantly different expectations about the capabilities to be provided could easily involve a requirements definition process that is five times more difficult than the corresponding process for a homogeneous group of users.

The complexity of requirements definition rises as a function of project duration. The longer a project goes on, the more likely it is that the software environment, customers, and end-users will change. A large application, whose total development will require several people working for a number of years, will involve a complex requirements definition process that does not terminate when design and implementation begin. Requirements changes will be requested throughout the development cycle and must be evaluated for their cost and schedule impact on the work already performed or underway.

It is difficult to identify the optimum effort to devote to requirements definition before undertaking software design. Determining this effort involves an assessment of the risk involved in assuming that the requirements are defined adequately to proceed. There will be a negative impact on the cost and schedule of subsequent life-cycle phases if all the requirements have not been identified or if they have not been stated with adequate precision.

III. Software Requirements Products

1. Results of requirements definition

The format in which the results of the requirements definition process should be presented depends upon the information needs of different audiences. End-users prefer a presentation that uses an application-oriented vocabulary, while software designers require more detail and a precise definition of application-specific terminology. However, requirements, no matter how presented, fall into four classes [Ross77b]:

- Functional requirements
- Non-functional requirements

- Inverse requirements
- Design and implementation constraints

a. Functional requirements

A *functional requirement* specifies a function that a system or system component (*i.e.*, software) must be capable of performing.

Functional requirements can be stated from either a static or dynamic perspective. The dynamic perspective describes the behavior of a system or system component in terms of the results produced by executing the system under specified circumstances. Functional requirements stated from an external, dynamic perspective are frequently written in terms of externally observable states; for example, the functions capable of being performed by an automobile cruise control system are different when the system is turned on from when it is disabled. Functional requirements stated from a static perspective describe the functions performed by each entity and the way each interacts with other entities and the environment.

b. Non-functional requirements

Non-functional requirements are those relating to performance, reliability, security, maintainability, availability, accuracy, error-handling, capacity, ability to be used by specific class of users, anticipated changes to be accommodated, acceptable level of training or support, or the like. They state characteristics of the system to be achieved that are not related to functionality. In a real-time system, performance requirements may be of critical importance, and functional requirements may need to be sacrificed in order to achieve minimally acceptable performance.

c. Inverse requirements (what the software *shall not* do)

Inverse requirements describe the constraints on allowable behavior. In many cases, it is easier to state that certain behavior must *never* occur than to state requirements guaranteeing acceptable behavior in all circumstances. Software safety and security requirements are frequently stated in this manner [Leveson86, Leveson87].

d. Design and implementation constraints

Design constraints and *implementation constraints* are boundary conditions on how the required software is to be constructed and implemented. They are givens of the development within which the designer must work. Examples of design constraints include the fact that the software must run using a certain database system or that the software must fit into the memory of a 512Kbyte machine.

2. Standards for requirements documents

The two most widely referenced standards relevant to producing requirements documents are U. S. Department of Defense Standard 2167A, *Military Standard for Defense System Software Development* [DoD88] and IEEE Standard 830, *IEEE Guide to Software Requirements Specifications* [IEEE84]. The IEEE standard describes the necessary content and qualities of a good requirements document and presents a recommended outline. Section 2 of the outline can be considered a template for customer/user-oriented requirements and section 3 a template for developer-oriented requirements. Even if a company standard for documentation format is to be used, the IEEE standard provides a good checklist of the items that should be included.

3. Customer/user-oriented software requirements

This section describes important characteristics of C-requirements products.

a. Objectives

C-requirements provide to the customer, who contracts for the software project and must accept the resulting software, a description of the functional requirements, non-functional requirements, inverse requirements, and design constraints adequate to commit to software development. "Adequate" means there is an acceptable level of risk regarding technical and cost feasibility and an acceptable level of risk regarding the completeness, correctness, and lack of ambiguity in the C-requirements. Acceptance criteria are usually developed in parallel with C-requirements.

b. Relative importance of specification attributes

Rombach describes the desirable attributes of specification products in general; the relative importance of these attributes depends upon the specification product. C-requirements must be understandable to the customer—and hence by end-users, who typically review the requirements before they are approved by the customer. They must therefore be written using the application vocabulary. Although understandability is the most important attribute of the C-requirements, there must be adequate precision for completeness, correctness, consistency, and freedom from ambiguity to be evaluated by analysts, users, and customers.

c. Key contents

The critical components of C-requirements are described below.

(i) Software functionality

Functionality and overall behavior of the software to be developed must be presented from a customer/user viewpoint. C-requirements can use a language other than natural English that allows the use of the application vocabulary. A prototype illustrating proposed software functionality may accompany C-requirements, but the conclusions drawn from the evaluation by customers and end-users should be stated explicitly.

(ii) Information definition and relationships

The information to be processed and stored, and the relationships between different types of information, must be defined. Entity-Relationship diagrams [Flavin81, Shlaer88] are frequently used for this purpose.

(iii) Critical non-functional requirements

(iv) Critical design constraints

(v) Acceptance criteria

4. Developer-oriented software requirements

D-requirements are usually produced by refining and augmenting the C-requirements. As an example, consider the description of the requirements for a scientific computation. The C-requirements might contain the equation to be solved and the numerical tolerance required, whereas the D-requirements would also contain the algorithm for solving the equation within the stated tolerance. During design, implementation of the specific algorithm would be chosen.

In many cases, an updated version of the C-requirements is developed during the creation of the D-requirements, as issues are resolved and more information is obtained from the customer, end-users, and "experts" in the application field. D-requirements products may exist at various levels of the software refinements process for the entire system, subsystem, or modules.

For a highly constrained system, where there is little requirements elicitation from people, only D-requirements are usually produced.

Important characteristics of D-requirements products are described below.

a. Objectives

D-requirements provide to the developer a description of the functional requirements, non-functional requirements, inverse requirements, and design constraints adequate to design and implement the software. "Adequate" means there is an acceptable level of risk regarding the completeness, consistency, and correctness of the information *for the purposes of software development*. Acceptance tests are usually developed in parallel with D-requirements.

C-requirements and D-requirements for embedded software are frequently combined into one document that is reviewed by technical representatives of the customer who have the expertise to review material whose principal audience is designers and implementors and who can verify the consistency of the C- with the D-requirements. The customer is usually much more concerned with the C-requirements for the total system. In cases where the requirements risks the customer will accept are very low—in a software system for airliner flight control, for example—a separate verification and validation contractor may be employed by the customer to verify, independently of the project team, that the D-requirements are consistent with the system requirements and are adequate to allow the team to proceed with software implementation.

b. Relative importance of specification attributes

D-requirements must be usable by designers and implementors without an in-depth knowledge of the application vocabulary and without direct contact with customers and end-users. Therefore, many aspects of the requirements must be more detailed than in the C-requirements, wherein the application vocabulary is expected to provide a common foundation of understanding among the customer and end-users. Application-specific information is frequently assumed by those who produce C-requirements, since it is inherent in understanding the terminology used. Precision in the D-requirements is essential, and less use of the application vocabulary—even at the cost of reduced understandability by application area experts—is usually required in order to achieve it.

c. Key contents

The critical components of D-requirements are described below.

(i) Software functionality

Functionality must be presented from the viewpoint of the software developer and must be sufficient in precision and detail for software design.

(ii) Information in C-requirements

No significant information appearing in the C-requirements may be omitted in preparing the D-requirements.

(iii) Interfaces to hardware/external systems

(iv) Critical non-functional requirements

(v) Critical design constraints

(vi) Acceptance criteria and acceptance tests

IV. Techniques and Tools for Performing Software Requirements Definition

The objective of this section is to introduce some of the techniques and computer support tools most likely to be used during requirements definition, but it is not intended to be a comprehensive description.

1. Techniques for eliciting requirements from people

Techniques used in a variety of fields for gathering information from people with different opinions (such as questionnaires and interviews) are relevant to defining software requirements. Davis [Davis83] and Powers [Powers84] cover most of the relevant methods.

In order to facilitate the elicitation of requirements, a variety of techniques have been developed that involve the participation of analysts, end-users, and customers in intensive working sessions over a period of several days. The objective is to speed up the negotiations between users with divergent opinions, to provide analysts with an in-depth understanding of software needs, and to complete a draft of the most important requirements. The analysts may develop models or prototypes during these sessions for review with the users. The best known of these techniques is Joint Application Development Technique (JAD), developed by IBM.

Frequent review of the work of analysts by customers and users facilitates agreement on requirements. An incremental process for reviewing models and accelerating the convergence of the requirements elicitation process has been formalized in the Reader-Author Cycle of the SADT methodology [Marca88]. The SADT approach is applicable to any model-building technique.

Walk-throughs [Freedman82] can be used to help determine the consistency and completeness of evolving requirements and to ensure that there is a common understanding among analysts, users, and customers of the implications of requirements. Yourdon [Yourdon89b] describes how to conduct walk-throughs using models built during requirements definition. Technical reviews [Collofello88b] can be utilized to assess the status of the requirements definition process.

2. Modeling techniques

Nearly all requirements definition techniques develop some type of model to structure the information gathered during requirements elicitation and to describe the functionality and behavior of software to meet the requirements. Most of the modeling methods are intended to be useful in a variety of application areas and are referred to here as "system modeling methods." For example, SADT [Ross85] has been applied to understanding how functions are performed manually in an organization and to building models showing the functions of a combined hardware/software system. However, there is also a role for other types of models in requirements definition, such as physical models (the layout of an assembly line to be automated) and simulation models (the actions proposed to take place on an automated assembly line).

Specification languages that are not graphically oriented have been proposed as an alternative to the graphically-oriented modeling languages widely used during requirements definition. None, however, has received significant usage for producing customer/user-oriented requirements and few have been used by other than their developers to produce developer-oriented requirements. One exception is the NRL/SCR requirements method [Heninger80], which is not graphically oriented and which has been applied to major projects.

Unfortunately, the developers of system modeling methods have used inconsistent terminology to describe their modeling approaches. It is usually difficult to understand what information can be represented easily using the modeling method and to what class of problems the approach is most applicable. White [White87] has done a thorough comparison of what can be represented using the most common model-building techniques. Pressman [Pressman87] surveys the following modeling methods and tools, which are among those described below: Structured Analysis, Real-Time Structured Analysis, Data Structured Systems Development, SADT, SREM/DCDS, and PSL/PSA. Davis [Davis88] surveys techniques for specifying the external behavior of systems and compares alternative approaches, including two formal specification languages.

A majority of modeling methods support describing a system in terms of several of the following characteristics:

- **Interfaces to external entities.** Since any model can describe only a well-defined subject area, the model-building notation must allow a precise description of what is to be included in the system of interest and how that system interfaces to external entities. In the case of a software system, the external entities typically are hardware, other software, and people. The ability to describe precisely the model interfaces is particularly important in requirements definition, since there may be divergent opinions among customers and users regarding the scope of the software to be developed in response to the requirements.

- **Functions to be performed.** All modeling methods widely used in requirements definition support the description of system functions, but they differ in how they describe the conditions under which functions are performed. For software that must react to external events (*i.e.*, real-time software), one must be able to describe precisely the events that cause a function to be performed.
- **Data Transformations.** Modeling methods that emphasize functions performing data transformations, such as Structured Analysis, are widely used in requirements definition for business data processing applications.
- **Structure of input/output data.** The structure of input and output data is modeled in requirements definition techniques that are designed to deal with complex information, such as Data Structured Systems Development (the Warnier-Orr methodology) [Orr81]. Typically, the structure of the information is assumed to be hierarchical. Such a model assists the analyst in understanding what items of information must be generated to produce a required report or screen display.
- **Relationships among information.** If the requirements indicate the software is to handle a significant number of items of information that are associated through complex relationships, information models can be used to show graphically the relationships between data obi^cts. The most widely used information modeling techniques [Flavin81] are Entity-Relationship (E-R) models and logical data models using the Curtice and Jones notation [Curtice82].
- **System behavior.** To model behavior as a system reacts to a sequence of externally-generated events requires representing the time sequence of inputs. Behavioral models are essential to the development of requirements for real-time systems.

3. Representative requirements definition methods

The following four groups of methods are the most frequently us?d in the United States. Each involves the production of a model for requirements representation. In Europe, the Jackson System Development method [Sutcliffe88] is also frequently used.

a. Structured Analysis and SADT

This group of methods emphasizes the graphic depiction of data transformations and functional decomposition. It incorporates the concept of a context diagram, which shows the external entities that provide information to the system or receive information from the system.

Structured Analysis is probably the most widely used graphically-oriented requirements definition technique. It is described in a number of books, including DeMarco [DeMarco79], Gane and Sarson [Gane79], and McMenamins and Palmer [McMenamins84]. The emphasis of the method is primarily on producing customer/user-oriented requirements.

SADT (Structured Analysis and Design Technique) [Ross85, Wallace87, Marca88] is a superset of Structured Analysis and was the first graphically-oriented method developed for use in performing requirements definition. Among its features are the use of interrelated multiple models to represent a system from the viewpoints of different participants in the requirements definition process [Leite88] and the ability to describe the states of data [Marca82]. A subset similar to Structured Analysis is known by the name IDEF. Also part of the method are procedures for conducting reviews of evolving models and team-oriented techniques for performing analysis and design. The emphasis of the method is primarily on producing customer/user-oriented requirements.

In Real-Time Structured Analysis, a state-diagrammatic representation is used to extend Structured Analysis to facilitate the description of system behavior. Alternative notations have been proposed by Hatley [Hatley87] and by Ward and Mellor [Ward85]. A consolidation of these two notations into a new notation called the Extended Systems Modeling Language has been proposed [Bruyn88]. An alternative notation for describing the states of a real-time system, Statecharts, has been developed by Harel [Harel88a].

b. DSSD

DSSD (Data Structured Software Development) (the Warnier-Ross Methodology) [Orr81] develops software requirements by focusing on the structure of input and output data. It assists the analyst in identifying key information objects and operations on those objects. The principal application of this graphically-oriented approach has been in the area of data processing systems.

c. SREM/DCDS

SREM (Software Engineering Requirements Methodology) [Alford77] was originally developed for performing requirements definition for very large embedded systems having stringent performance requirements. With the addition of extensions to support distributed concurrent systems, the name has been changed to the Distributed Computer Design System [Alford85]. The emphasis of SREM/DCDS is primarily on producing developer-oriented requirements.

d. NRL/SCR

The NRL/SCR (Naval Research Laboratory Software Cost Reduction) requirements method [Heninger80] is oriented toward embedded systems and produces developer-oriented requirements. It differs from the methods listed above by being a "black box" requirements method, in which requirements are stated in terms of input and output data items and externally visible characteristics of the system state. The method is intended to separate clearly design issues from requirements issues, and it is sufficiently different in its assumptions from the other methods that it is worthy of detailed study. The work of Mills [Mills86] is based on similar assumptions.

4. Computer support tools for model development

Tools for use on personal computers and workstations to support the most widely used modeling methods are evolving rapidly, and published information on available tools is outdated within a few months of publication. The best sources of current information are the exhibitions associated with major conferences such as the International Conference on Software Engineering and CASExpo.

a. Method-specific tools

Computer support tools designed for notations used by specific methods are commercially available for all the modeling methods listed above except SREM/DCDS and NRL/SCR. Tools are also available to support the development of information models using both the ERA notation and the Curtice and Jones notation.

b. Non-method-specific tools

Tools that are not specific to the notation of a particular modeling method fall into two categories: tools that can be user-customized to represent the notation, objects, and relationships specific to a given modeling method; and tools that require a translation between the notation of the modeling method and the notation required by the tool. PSL/PSA (Problem Statement Language/Problem Statement Analyzer) [Teichroew77], which was the first widely available computer tool to support requirements analysis, is in the second category.

5. Computer support tools for prototyping

Available computer tools to support prototyping are rapidly increasing in capability. For user interface prototyping on personal computers, widely used tools are Hypercard on the Apple Macintosh and Dan Bricklin's Demo II Program on the IBM PC. Statemate [Harel88b] supports the development on a workstation of a combined user interface prototype and an essential functionality prototype through the use of an executable model that describes the functionality and behavior of the system

Teaching Considerations

Suggested Course Types

The material presented in this module is intended to be used in one of three ways:

1. As background material for teachers preparing software engineering courses.
2. As material for a course containing a series of lectures on software requirements.
3. As material for a teacher planning a software requirements definition project course.

The author is currently using the module material to teach a course, *Software System Design*, in the software engineering master's degree program at Boston University. Of the 26 lectures, 10 are on system and software requirements, and 16 are on architectural design.

While on the faculty of the Wang Institute, the author supervised project courses in which teams of 4-5 students performed requirements definition projects for external customers [Brackett88].

Teaching Experience

In the author's experience, it is difficult, if not impossible, to convey adequately the principal concepts in this module without having the students undertake some type of requirements definition project. Nearly all students, even those with 3-5 years of industrial experience, lack *any* experience in performing requirements definition. Therefore, survey courses are likely only to introduce the student to the need to specify software requirements and to some of the modeling and prototyping techniques frequently used.

A requirements definition project requires adequate calendar time for the student to produce a C-requirements document, the teacher to provide detailed feedback on it, and the student to prepare a second (or third!) iteration. Each iteration should define the requirements more completely and precisely, while reducing the number of design solutions the students identify as requirements. Following the completion of C-requirements, D-requirements should be developed, if time permits. Because most students have no experience in dealing with customer and end-user issues, they learn most by producing C-requirements.

The author's software system design course uses *IEEE Guide for Software Requirement Specifications* [IEEE84] to define the outline and content of the C-requirements document. Techniques for requirements definition must be taught in enough detail for students to apply them. In practice, this means emphasizing one technique for each step of the process, even though it would desirable to expose the student to a wide variety of techniques. The author currently teaches the details of

- Information modeling using the Curtice and Jones notation [Curtice82].
- Real-Time Structured Analysis [Hatley-87].
- Support tools associated with the above.

Since producing models is not the principal objective of requirements definition, emphasis in the course and in the instructor's review of the project documents must be given to non-functional requirements, the handling of unexpected events, and design constraints.

The following projects have been used in teaching *Software System Design*:

- The requirements for the first automatic teller machine (assuming the project was conducted in 1977).
- The requirements for the software to control 5 elevators in a 50-story building.

As the major assignment in the course, each has been adequately done to the C-requirements level by teams of three to four students in about four weeks. To continue to the D-requirements level would require about an additional four weeks.

Suggested Reading Lists

The following lists categorize items in the bibliography by applicability.

Instructor Essential: This is a small set of readings intended to provide, in conjunction with this module, the basic information an instructor needs to prepare a series of lectures on software requirements.

Instructor Recommended: These readings provide the instructor with additional material and, if time permits, should be reviewed in conjunction with the *Instructor Essential* items.

Detailed: These readings have been included to provide access to the literature on specific topics or to materials that are secondary sources to those listed under the *Instructor Essential* or *Instructor Recommended* categories.

* Those readings marked with "*" are suitable for use by students in a graduate-level course including a series of lectures on software requirements.

† Possible textbooks are indicated with "†".

No single book suitable both for an information-systems–oriented course and a real-time–oriented course can be recommended.

Paper Categories

Instructor Essential

- Davis88
- IEEE84*
- *one of* { Martin88* / Powers84*† }
- *one of* { Pressman87*† / Sommerville89*† }

Instructor Recommended

- Boehm84a*
- Brooks87*
- Clapp87*
- Flavin81*
- *one of* { Hatley87*† / Ward85*† }
- Heninger80
- Rombach90
- Ross77b*
- Yourdon89a

Detailed

- Abbott86
- Alford77
- Alford85
- Boehm84b*
- Boehm86*
- Brackett88
- Bruyn88
- Collofello88a
- Collofello88b
- Curtice82*
- Davis82*
- Davis83*†
- DeMarco79*
- DoD88
- Freedman82
- Gane79*
- Gause89
- Gomaa81*
- Gomaa89*
- Harel88a
- Harel88b*

Detailed (cont.)

- IEEE83
- Kowall88*†
- Leite87
- Leite88
- Leveson86*
- Leveson87
- Marca82
- Marca88*
- McMenamins84*
- Mills86
- Orr81
- Perlman88
- Ross85
- Shlaer88
- Sutcliffe88
- Teichroew77*
- Wallace87
- Ward89
- White87
- Yourdon89b

Bibliography

Abbott86

Abbott, R. J. *An Integrated Approach to Software Development.* New York: John Wiley, 1986.

Table of Contents
1 Introduction

PART 1: REQUIREMENTS
2 Requirements Discussion
3 Requirements Document Outline

PART 2: SYSTEM SPECIFICATION
4 Discussion
5 Behavioral Specification Outline
6 Procedures Manual
7 Administrative Manual

PART 3: DESIGN
8 Design Discussion
9 System Design Documentation
10 Component Documentation: Specification and Design

Appendix: Abstraction and Specification
References
Index

This is a general software engineering text, organized as a collection of annotated outlines for technical documents important to the development and maintenance of software. The outline of Abbott's requirements document differs from [IEEE84], and the instructor may find it useful to compare the differences. The process of requirements definition is not explained in detail, so this book is not an adequate stand-alone text for a series of lectures on software requirements.

Alford77

Alford, M. "A Requirements Engineering Methodology for Real-Time Processing Requirements." *IEEE Trans. Software Eng. SE-3*, 1 (Jan. 1977), 60-69.

Abstract: This paper describes a methodology for the generation of software requirements for large, real-time unmanned weapons systems. It describes what needs to be done, how to evaluate the intermediate products, and how to use automated aids to improve the quality of the product. An example is provided to illustrate the methodology steps and their products and the benefits. The results of some experimental applications are summarized.

This paper should be read in conjunction with [Alford85] and [Davis88].

Alford85

Alford, M. "SREM at the Age of Eight: The Distributed Computing Design System." *Computer 18*, 4 (April 1985), 36-46.

SREM/DCDS has been used primarily on very large government contracts, but the supporting tools make it unsuitable for use in an academic course. They are difficult to learn and somewhat difficult to install.

Boehm84a

Boehm, B. W. "Verifying and Validating Software Requirements and Design Specifications." *IEEE Software 1*, 1 (Jan. 1984), 75-88.

An excellent survey article, which is understandable by students.

Boehm84b

Boehm, B. W., T. E. Gray, and T. Seewaldt. "Prototyping vs. Specifying: A Multi-Project Experiment." *Proc. 7th Intl. Conf. Software Eng.* New York: IEEE, 1984, 473-484.

Abstract: In this experiment, seven software teams developed versions of the same small-size (2000-4000 source instruction) application software product. Four teams used the Specifying approach. Three teams used the Prototyping approach.

The main results of the experiment were:

Prototyping yielded products with roughly equivalent performance, but with about 40% less code and 45% less effort.

The prototyped products rated somewhat lower on functionality and robustness, but higher on ease of use and ease of learning.

Specifying produced more coherent designs and software that was easier to integrate.

The paper presents the experimental data supporting these and a number of additional conclusions.

Boehm86

Boehm, B.W. "A Spiral Model of Software Development and Enhancement." *ACM Software Engineering Notes 11*, 4 (Aug. 1986), 14-24.

This paper, reprinted from the proceedings of the March 1985 International Workshop on the Software Process and Software Environments, presents Boehm's spiral model. The author's description from the introduction:

The spiral model of software development and enhancement presented here provides a new framework for guiding the software process. Its major distinguishing feature is that it creates a *risk-driven* approach to the software process, rather than a strictly specification-driven or prototype-driven process. It incorporates many of the strengths of other models, while resolving many of their difficulties.

Brackett88

Brackett, J. W. "Performing Requirements Analysis Project Courses for External Customers." In *Issues in Software Engineering Education*, R. Fairley and P. Freeman, eds. New York: Springer-Verlag, 1988, 56-63.

Brooks87

Brooks, F. "No Silver Bullet: Essence and Accidents of Software Engineering." *Computer 20*, 4 (April 1987), 10-19.

Bruyn88

Bruyn, W., R. Jensen, D. Keskar, and P. T. Ward. "ESML: An Extended Systems Modeling Language Based on the Data Flow Diagram." *ACM Software Engineering Notes 13*, 1 (Jan. 1988), 58-67.

Abstract: ESML (Extended Systems Modeling Language) is a new system modeling language based on the Ward-Mellor and Boeing structured methods techniques, both of which have proposed certain extensions of the DeMarco data flow diagram notation to capture control and timing information. The combined notation has a broad range of mechanisms for describing both combinatorial and sequential control logic.

This paper should be read in conjunction with [Ward89].

Clapp87

Clapp, J. "Rapid Prototyping for Risk Management." *Proc. COMPSAC 87*. Washington, D. C.: IEEE Computer Society Press, 1987, 17-22.

Abstract: Rapid prototyping is useful for controlling risks in the development and upgrade of decision support systems. These risks derive from uncertainty about what the system should do, how its capabilities should be achieved, how much it will cost, and how long it will take to complete. This paper describes uses of rapid prototyping for risk management and summarizes lessons learned from their use....

Collofello88a

Collofello, J. S. *Introduction to Software Verification and Validation*. Curriculum Module SEI-CM-13-1.1, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., Dec. 1988.

Capsule Description: Software verification and validation techniques are introduced and their applicability discussed. Approaches to integrating these techniques into comprehensive verification and validation plans are also addressed. This curriculum module provides an overview needed to understand in-depth curriculum modules in the verification and validation area.

Collofello88b

Collofello, J. S. *The Software Technical Review Process*. Curriculum Module SEI-CM-3-1.5, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., June 1988.

Capsule Description: This module consists of a comprehensive examination of the technical review process in the software development and maintenance life cycle. Formal review methodologies are analyzed in detail from the perspective of the review participants, project management and software quality assurance. Sample review agendas are also presented for common types of reviews. The objective of the module is to provide the student with the information necessary to plan and execute highly efficient and cost effective technical reviews.

Curtice82

Curtice, R. and P. Jones. *Logical Data Base Design*. New York: Van Nostrand Reinhold, 1982.

This book introduces a data modeling notation that is easily taught to students and that facilitates decomposing a large data model into smaller submodels. However, the text is not oriented toward using data models during requirements definition.

Davis82

Davis, G. B. "Strategies for Information Requirements Determination." *IBM Systems J. 21*, 1 (1982), 4-30.

Abstract: Correct and complete information requirements are key ingredients in planning organizational information systems and in implementing information system applications. Yet, there has been relatively little research on information requirements determination, and there are relatively few practical, well-formulated procedures for obtaining complete, correct information requirements. Methods for obtaining and documenting information requirements are proposed, but they tend to be presented as general solutions rather than alternative methods for implementing a chosen strategy of requirements determination. This paper identifies two major levels of requirements: the organizational information requirements reflected in a

planned portfolio of applications and the detailed information requirements to be implemented in a specific application. The constraints on humans as information processors are described in order to explain why "asking" users for information requirements may not yield a complete, correct set. Various strategies for obtaining information requirements are explained. Examples are given of methods that fit each strategy. A contingency approach is then presented for selecting an information requirements determination strategy. The contingency approach is explained both for defining organizational information requirements and for defining specific, detailed requirements in the development of an application.

Davis83

Davis, W. S. *Systems Analysis and Design.* Reading, Mass.: Addison-Wesley, 1983.

Table of Contents

I. THE SYSTEM DEVELOPMENT PROCESS
1 *Structured Systems Analysis and Design*
2 *Case A: Problem Definition*
3 *Case A. The Feasibility Study*
4 *Case A: Analysis*
5 *Case A: System Design*
6 *Case A: Detailed Design*
7 *Case A: Implementation and Maintenance*

II. A SMALL BUSINESS SYSTEM
8 *Case B: Problem Definition*
9 *Case B: The Feasibility Study*
10 *Case B: Analysis*
11 *Case B: System Design*
12 *Case B: Detailed Design*
13 *Case B: Implementation and Maintenance*

III. AN ON-LINE SYSTEM
14 *Case C: Problem Definition*
15 *Case C: The Feasibility Study*
16 *Case C: Analysis*
17 *Case C: System Design*
18 *Case C: Detailed Design*
19 *Case C: Implementation and Maintenance*

IV. THE ANALYST'S TOOLS
Module A: Inspections and Walkthroughs
Module B: Interviewing
Module C: The Feasibility Study
Module D: Data Flow Diagrams
Module E: Data Dictionaries
Module F: System Flowcharts
Module G: Cost/Benefit Analysis
Module H: HIPO with Structured English
Module I: Pseudocode
Module J: Program Logic Flowcharts
Module K: Warnier/Orr Diagrams
Module L: PERT and CPM
Module M: File Design and Space Estimates
Module N: Forms Design and Report Design
Module O: Decision Tables and Decision Trees

A text for a first undergraduate course in analysis and design, based on three case studies. Each of the case studies is taken through the steps of problem definition, feasibility study, analysis, system design, detailed design. The main emphasis of the book is on analysis rather than design. The book is oriented toward business applications and primarily makes use of Structured Analysis and Structured Design. The case studies may provide a useful basis for class discussions.

Davis88

Davis, A. "A Comparison of Techniques for the Specification of External System Behavior." *Comm. ACM 31*, 9 (Sept. 1988), 1098-1115.

This paper compares finite state techniques, decision tables, program design language, Real-Time Structured Analysis, statecharts, REVS/SREM, Petri nets, and three languages: SDL (Specification and Description Language), RLP (Requirements Language Processor), and PAISLey. It is the best survey paper in the area of notation and tools supporting requirements definition, and it includes an extensive bibliography.

DeMarco79

DeMarco, T. *Structured Analysis and System Specification.* Englewood Cliffs, N. J.: Yourdon Press, 1979. Also published by Prentice-Hall, 1979.

A very readable book on Structured Analysis and system specification that covers data flow diagrams, data dictionaries, and process specification. However, Structured Analysis has evolved greatly since 1979, and [Yourdon89a] is a more up-to-date reference.

DoD88

DoD. *Military Standard for Defense System Software Development.* DOD-STD-2167A, U. S. Department of Defense, Washington, D.C., 29 February 1988.

Flavin81

Flavin, M. *Fundamental Concepts of Information Modeling.* Englewood Cliffs, N. J.: Yourdon Press, 1981.

A well-written book that is a good introduction to information (data) modeling for the instructor. Flavin describes this short work in the preface, part of which is reproduced here:

> Information modeling is a modern form of system analysis that identifies the objects, relationships, and operations composing some real-world system.

It is used for database design and business system analysis and planning.

As an analytical procedure, it is composed of two major parts: an analytical "front end," and a representational "back-end." The analytical front-end is a coherent set of procedures for finding, identifying, and defining objects; relationships; operations that modify the objects and relationships; and data elements that describe objects and relationships. The representational back-end is a set of procedures for mapping the semantic components of the model onto data structures that represent and describe each component. Information modeling is a marriage of the art of system analysis with the science of data representation.

This monograph is intended to make the system analysis component less artistic and a bit more scientific, and to lay out a proper conceptual foundation for the construction of an Entity-Relationship (E-R) model of some real-world system.

Freedman82

Freedman, D. P., and G. M. Weinberg. *Handbook of Walkthroughs, Inspections, and Technical Reviews: Evaluating Programs, Projects, and Products, 3rd Ed.* Boston: Little, Brown, 1982.

This book is a secondary source to [Yourdon89b].

Gane79

Gane, C., and T. Sarson. *Structured Systems Analysis: Tools and Techniques.* Englewood Cliffs, N. J.: Prentice-Hall, 1979.

One of the more widely used books on Structured Analysis. The book discusses some of the problems in analysis, reviews graphical tools, and shows how the graphical tools fit together to make a logical model. Although this book is very readable, Structured Analysis has evolved greatly since 1979, and [Yourdon89a] is a more up-to-date reference.

Gause89

Gause, D. C., and G. M. Weinberg. *Exploring Requirements: Quality Before Design.* New York: Dorset House, 1989.

From the publisher:

The authors focus on three critical but neglected human aspects of the requirements process: developing a consistent understanding of requirements among all participants, encouraging the desire to work as a team on the project, and creating the necessary skills and tools for working effectively as a team to define requirements. Topics include ambiguity and ambiguity metrics; techniques for generating ideas; right-brain methods; choosing project names; conflict resolution; attributes, functions, and constraints; expectations; reaching agreements; and user satisfaction tests.

Gomaa81

Gomaa, H. and D. B. H. Scott. "Prototyping as a Tool in the Specification of User Requirements." *Proc. 5th Intl. Conf. Software Eng.* New York: IEEE, 1981, 333-339.

Abstract: One of the major problems in developing new computer applications is specifying the user's requirements such that the requirements specification is correct, complete, and unambiguous. Although prototyping is often considered too expensive, correcting ambiguities and misunderstandings at the specification stage is significantly cheaper than correcting a system after it has gone into production. This paper describes how a prototype was used to help specify the requirements of a computer system to manage and control a semiconductor processing facility. The cost of developing and running the prototype was less than 10% of the total software development cost.

This is an excellent case study that is suitable for study by students.

Gomaa89

Gomaa, H. *Software Design Methods for Real-Time Systems.* Curriculum Module SEI-CM-22-1.0, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., Dec. 1989.

Capsule Description: This module describes the concepts and methods used in the software design of real-time systems. It outlines the characteristics of real-time systems, describes the role of software design in real-time system development, surveys and compares some software design methods for real-time systems, and outlines techniques for the verification and validation of real-time designs. For each design method treated, its emphasis, concepts on which it is based, steps used in its application, and an assessment of the method are provided.

Harel88a

Harel, D. "On Visual Formalisms." *Comm. ACM 31,* 5 (May 1988), 514-530.

An elegant and clearly written paper which discusses a number of important issues about model representation. While the first part of the paper is concerned with general issues, the latter part provides an interesting exposition of *statecharts,* and includes a detailed example in the form of a description of a digital watch. The paper will be of particular interest to instructors concerned with the imprecision of the graphic notations frequently used to describe software requirements. It should be read in conjunction with [Ward89] and [Davis88].

Harel88b

Harel, D., *et al.* "STATEMATE: A Working Environment for the Development of Complex Reactive Systems." *Proc. 10th Intl. Conf. on Software Eng.* Washington, D. C.: IEEE Computer Society Press, 1988, 396-406.

Abstract: This paper provides a brief overview of the STATEMATE system, constructed over the past three years by i-Logix Inc., and Ad Cad Ltd. STATEMATE is a graphical working environment, intended for the specification, analysis, design and documentation of large and complex reactive systems, such as real-time embedded systems, control and communication systems, and interactive software. It enables a user to prepare, analyze and debug diagrammatic, yet precise, descriptions of the system under development from three inter-related points of view, capturing structure, functionality *and* behavior. *These views are represented by three graphical languages, the most intricate of which is the language of* statecharts *used to depict reactive behavior over time. In addition to the use of statecharts, the main novelty of STATEMATE is in the fact that it 'understands' the entire descriptions perfectly, to the point of being able to analyze them for crucial dynamic properties, to carry out rigorous animated executions and simulations of the described system, and to create runing code automatically. These features are invaluable when it comes to the quality and reliability of the final outcome.*

Hatley87

Hatley, D. J., and I. A. Pirbhai. *Strategies for Real-Time System Specification.* New York: Dorset House, 1987.

Table of Contents

PART I: THE OVERALL STRATEGY
1 Overview
2 The Role of the Methods

PART II: THE REQUIREMENTS MODEL
3 Overview
4 The Process Model
5 The Control Model
6 Finite State Machines
7 Timing Requirements
8 Requirements Dictionary
9 Requirements Model Interpretation and Summary

PART III: BUILDING THE REQUIREMENTS MODEL
10 Overview
11 Getting Started
12 Developing the Model's Structure
13 Preparing Process Specifications
14 Preparing Control Specifications
15 Defining Timing
16 Managing the Dictionary

PART IV: THE ARCHITECTURE MODEL
17 Overview
18 Architecture Diagrams
19 Architecture Dictionary and Module Specifications
20 Completing the Architecture Model

PART V: BUILDING THE ARCHITECTURE MODEL
21 Overview
22 Enhancing the Requirements Model
23 Creating the System Architecture Model
24 Creating the Hardware and Software Architecture Models
25 Architecture Development Summary

PART VI: EXAMPLES
26 Automobile Management System
27 Home Heating System
28 Vending Machine

Appendix A: Standard Symbols and Definitions
Appendix B: Making the Models into Documents
Appendix C: Information Modeling—The Third Perspective

This is a well-written text on Real-Time Structured Analysis. This book should be read in conjunction with [Ward89] in order better to understand the capabilities of the notation. This text and [Ward85] are alternative texts; the choice of a text for teaching Real-Time Structured Analysis may depend upon whether the computer tools to be used support only the Hatley notation or only the Ward notation.

Heninger80

Heninger, K. L. "Specifying Software Requirements for Complex Systems: New Techniques and Their Applications." *IEEE Trans. Software Eng. SE-6*, 1 (Jan. 1980), 2-13.

Abstract: This paper concerns new techniques for making requirements specifications precise, concise, unambiguous, and easy to check for completeness and consistency. The techniques are well-suited for complex real-time software systems; they were developed to document the requirements of existing flight software for the Navy's A-7 aircraft. The paper outlines the information that belongs in a requirements document and discusses the objectives behind the techniques. Each technique is described and illustrated with examples from the A-7 document. The purpose of the paper is to introduce the A-7 document as a model of a disciplined approach to requirements specification; the document is available to anyone who wishes to see a fully worked out example of the approach.

This paper shows how software requirements can be defined using a description of external system behavior. The technique is part of the requirements and design methodology developed at the Naval Research Laboratory by Parnas, Clements and Weiss [Gomaa89]. The approach should be reviewed by the instructor, since it does not use a graphic model of system functionality; it is based upon different assumptions about how to best describe software requirements.

IEEE83

IEEE. *IEEE Standard Glossary of Software Engineering Terminology.* New York: IEEE, 1983. ANSI/IEEE Std 729-1983.

This standard provides definitions for many of the terms used in software engineering.

IEEE84

IEEE. *IEEE Guide to Software Requirements Specifications.* New York: IEEE, 1984. ANSI/IEEE Std 830-1984.

An excellent description of the contents of a software requirements document.

Kowall88

Kowall, J. *Analyzing Systems.* Englewood Cliffs, N. J.: Prentice-Hall, 1988.

Table of Contents

PART I: PHYSICAL SPECIFICATIONS
1 *Introduction*
2 *Data Flow Diagrams*
3 *Data Dictionary*
4 *Mini-Specifications*
5 *Physical Models*

PART II: LOGICAL SPECIFICATIONS
6 *Logical Analysis*
7 *Object Analysis*
8 *Event Analysis*
9 *Logical Models*

PART III: REAL-TIME MODELS AND SYSTEM ARCHITECTURE
10 *Real-Time Systems Specifications*
11 *Convenient Auto Rental System*
12 *Systems Architecture*

Glossary
References
Index

This well-written book is a potential text for a course in which a substantial amount of time is devoted to requirements analysis. It includes Structured Analysis, including an introduction to the real-time extensions, and information modeling. The chapters on event analysis and object analysis, plus the 50-page car rental system example, will be of interest to the instructor even if the book is not used as a text.

Leite87

Leite, J. *A Survey on Requirements Analysis.* RTP 071, University of California, Irvine, June 1987.

This report contains an excellent annotated bibliography.

Leite88

Leite, J. *Viewpoint Resolution in Requirements Elicitation.* Ph.D. Th., University of California, Irvine, 1988. Available from University Microfilms International, Ann Arbor, Michigan.

A valuable thesis to anyone working seriously in developing requirements definition methods.

Leveson86

Leveson, N. G. "Software Safety: Why, What, and How." *ACM Computing Surveys 18*, 2 (June 1986), 125-163.

Software safety requirements analysis is described in detail here. This survey contains a very long bibliography at the end to aid in finding further information.

Leveson87

Leveson, N. G. *Software Safety.* Curriculum Module SEI-CM-6-1.0, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., April 1987.

***Capsule Description:** Software safety involves ensuring that software will execute within a system context without resulting in unacceptable risk. Building safety-critical software requires special procedures to be used in all phases of the software development process. This module introduces the problems involved in building such software along with the procedures that can be used to enhance the safety of the resulting software product.*

Marca82

Marca, D. A., and C. L. McGowan. "Static and Dynamic Data Modeling for Information System Design." *Proc. 6th Intl. Conf. on Software Eng.* New York: IEEE, 1982, 137-146.

This paper shows how Entity-Relationship models and SADT data models can both be used during requirements definition.

Marca88
Marca, D. A., and C. L. McGowan. *SADT: Structured Analysis and Design Technique.* New York: McGraw-Hill, 1988.

A detailed description of SADT, the predecessor to Structured Analysis. The book makes use of a generous supply of illustrations and examples, as well as providing a number of case studies taken from different application domains. The large-size format used for the book makes the examples particularly clear and readable.

The level of detail provided makes this particularly suitable for use as a source of material for the instructor. It should be read in conjunction with [Ross85].

Martin88
Martin, C. *User-Centered Requirements Analysis.* Englewood Cliffs, N.J.: Prentice-Hall, 1988.

A well-written book emphasizing the process of requirements definition for information systems, for which a majority of requirements are gathered from people. The chapters on performance requirements and on "objectives analysis," which covers the context analysis process, are of particular interest.

McMenamins84
McMenamins, S. M, and J. F. Palmer. *Essential Systems Analysis.* New York: Yourdon Press, 1984.

One of the best books on the process of performing requirements definition using Structured Analysis.

Mills86
Mills, H. D., C. Linger, and A. R. Hevner. *Principles of Information Systems Analysis and Design.* Orlando, Fla.: Academic Press, 1986.

This book describes an approach to requirements definition for information systems that emphasizes the use of models showing external system behavior. Black-box and state-machine models are used; these are similar in concept to the form of representation described in [Heninger80].

Orr81
Orr, K. *Structured Requirements Definition.* Topeka, Kan.: Ken Orr and Associates, 1981. Available from Optima, Inc., Schaumburg, Ill.

This book describes a methodology and notation for performing requirements definition for information systems. The technique is a data-structured approach that focuses on the data that the system will transform into information. The approach is considerably different in its strategic approach from Structured Analysis. Orr's work is worthy of study by the instructor, since it enjoys significant industrial usage.

Perlman88
Perlman, G. *User Interface Development.* Curriculum Module SEI-CM-17-1.0, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., April 1988.

***Capsule Description:** This module covers the issues, information sources, and methods used in the design, implementation, and evaluation of* user interfaces, *the parts of software systems designed to interact with people. User interface* design *draws on the experiences of designers, current trends in input/output technology, cognitive psychology, human factors (ergonomics) research, guidelines and standards, and on the feedback from evaluating working systems. User interface* implementation *applies modern software development techniques to building user interfaces. User interface* evaluation *can be based on empirical evaluation of working systems or on the predictive evaluation of system design specifications.*

Powers84
Powers, M., D. Adams, and H. Mills. *Computer Information Systems Development: Analysis and Design.* Cincinnati: South-Western, 1984.

Table of Contents
I. OVERVIEW
1 *The Systems Development Environment*
2 *The Systems Development Life Cycle*

II. THE INVESTIGATION PHASE
3 *Initial Investigation*
4 *Information Gathering*
5 *Feasibility Study*
6 *The Process and Products of Analysis*
7 *Cost/Benefit Analysis*
8 *Communication*

III. ANALYSIS AND GENERAL DESIGN PHASE
9 *Existing System Review*
10 *System Modeling Tools*
11 *New System Requirements*
12 *Output Design*
13 *Input Design*
14 *Logical Data Analysis*
15 *New System Design*
16 *File Design*
17 *Control and Reliability Design*
18 *Implementation and Installation Planning*

IV. IMPLEMENTATION, INSTALLATION, AND RE VIEW PHASES
19 *Detailed Design and Implementation Phase*
20 *Installation*

21 Review
22 Project Management

Appendix A: Systems Analysis Project
Appendix B: A Case Scenario
Glossary
Index

A possible text for teaching requirements definition for information systems to undergraduates. The section "The Investigation Phase" covers requirements identification and requirements analysis in detail.

Pressman87

Pressman, R. S. *Software Engineering: A Practitioner's Approach, 2nd Ed.* New York: McGraw-Hill, 1987.

Table of Contents

1 Software and Software Engineering
2 Computer System Engineering
3 Software Project Planning
4 Requirements Analysis Fundamentals
5 Requirements Analysis Methods
6 Software Design Fundamentals
7 Data Flow-Oriented Design
8 Data Structure-Oriented Design
9 Object-Oriented Design
10 Real-Time Design
11 Programming Languages and Coding
12 Software Quality Assurance
13 Software Testing Techniques
14 Software Testing Strategies
15 Software Maintenance and Configuration Management

Epilogue
Index

A suitable text for a graduate-level course in software engineering.

Rombach90

Rombach, H. D. *Software Specification: A Framework.* Curriculum Module SEI-CM-11-2.1, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pa., Jan. 1990.

Capsule Description: This curriculum module presents a framework for understanding software product and process specifications. An unusual approach has been chosen in order to address all aspects related to "specification" without confusing the many existing uses of the term. In this module, the term specification *refers to any plan (or standard) according to which products of some type are constructed or processes of some type are performed, not to the products or processes themselves. In this sense, a specification is itself a product that describes how products of some type should look or how processes of some type should be performed. The framework includes:*

- *A reference software life-cycle model and terminology*
- *A characterization scheme for software product and process specifications*
- *Guidelines for using the characterization scheme to identify clearly certain life-cycle phases*
- *Guidelines for using the characterization scheme to select and evaluate specification techniques*

Ross77a

Special Collection on Requirements Analysis. Ross, D. T., ed. *IEEE Trans. Software Eng. SE-3*, 1 (Jan. 1977).

This special journal issue contains [Ross77b], [Teichroew77], and [Alford77], plus other papers on requirements methods.

Ross77b

Ross, D. T., and K. E. Schoman, Jr. "Structured Analysis for Requirements Definition." *IEEE Trans. Software Eng. SE-3*, 1 (Jan. 1977), 6-15.

Abstract: Requirements definition encompasses all aspects of system development prior to actual system design. We see the lack of an adequate approach to requirements definition as the source of major difficulties in current systems work. This paper examines the needs for requirements definition, and proposes meeting those objectives with three interrelated subjects: context analysis, functional specification, and design constraints. Requirements definition replaces the widely used, but never well-defined, term "requirements analysis."

The purpose of this paper is to present, in a comprehensive manner, concepts that apply throughout requirements definition (and, by implication, to all of system development). The paper discusses the functional architecture of systems, the characteristics of good requirements documentation, the personnel involved in the process of analysis, and management guidelines that are effective even in complex environments.

The paper then outlines a systematic methodology that incorporates, in both notation and technique, the concepts previously introduced. Reference is made to actual requirements definition experience and to practicable automated support tools that may be used with the methodology.

This paper was the first publication on SADT [Marca88, Ross85], but its principal value is its description of the requirements definition process:

Ross85

Ross, D. T. "Applications and Extensions of SADT." *Computer 18*, 4 (April 1985), 25-34.

Rzepka85

Special Issue on Requirements Engineering Environments. W. Rzepka and Y. Ohno, eds. *Computer 18*, 4 (April 1985).

This journal issue contains [Ross85] and [Alford85], plus other papers on requirements methods for real-time applications.

Shlaer88

Shlaer, S., and S. J. Mellor. *Object-Oriented Systems Analysis: Modeling the World in Data.* Englewood Cliffs, N. J.: Yourdon Press, 1988.

Sommerville89

Sommerville, I. *Software Engineering, 3rd Ed.* Wokingham, England: Addison-Wesley, 1989.

This latest edition of Sommerville's text expands upon the excellent chapter on requirements definition in the 2nd edition. Topics include system (including data) modeling, non-functional requirements definition, and requirements validation.

Sutcliffe88

Sutcliffe, A. *Jackson System Development.* New York: Prentice-Hall, 1988.

From the introductory chapter·

> [Jackson System Development (JSD)] is organized in three separate stages which guide the analyst through the systems development process. Each stage has a set of activities with clear start and end points (this helps the analyst using the method) and facilitates project control as deliverables can be defined for each stage. The three stages can be outlined briefly as follows.
>
> (a) *Modelling stage.* A description is made of the real world problem and the important actions within the system are identified. This is followed by analysis of the major structures within the system, called *entities* in JSD. . . .
>
> (b) *Network stage.* The system is developed as a series of subsystems. First the major structures are taken from the modelling stage and input and outputs are added; this is followed by the analysis of the output subsystem which provides information, and then of the input subsystem which handles the user interface and validation. . . .
>
> (c) *Implementation stage.* In this stage the logical system specification, which is viewed as a network of concurrently communicating processes, is transformed into a sequential design by the technique of scheduling. This is followed by further detailed design and coding. . . .
>
> JSD begins by analysing the major system structures which are important to create a model of the system problem, the entities. Then these structures are connected together to create a network model of the system, while at the same time the design is elaborated by addition of other processes to create output, and to handle input messages and user interaction. The essence . . . is to create a system model of reality first and then to add the functionality.

JSD is usually not considered to support requirements definition, but Jackson's emphasis on modeling the problem domain makes it a viable alternative, for information systems, to functional, top-down approaches such as Structured Analysis. This book is unique in showing how JSD relates to more widely used software requirements and design techniques. [Ward89] also shows how its notation relates to more widely used requirements notations.

Teichroew77

Teichrow, D., and E. A. Hershey, III. "PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems." *IEEE Trans. Software Eng. SE-3*, 1 (Jan. 1977), 41-48.

Abstract: PSL/PSA is a computer-aided structured documentation and analysis technique that was developed for, and is being used for, analysis and documentation of requirements and preparation of functional specifications for information processing systems. The present status of requirements definition is outlined as the basis for describing the problem which PSL/PSA is intended to solve. The basic concepts of the Problem Statement Language are introduced and the content and use of a number of standard reports that can be produced by the Problem Statement Analyzer are briefly described.

The experience to date indicates that computer-aided methods can be used to aid system development during the requirements definition stage and that the main factors holding back such use are not so much related to the particular characteristics and capabilities of PSL/PSA as they are to organizational considerations involved in any change in methodology and procedure.

PSL/PSA was the first widely used computer tool to support requirements definition. Although its use is declining, its objectives are relevant to the design of newer CASE tools.

Wallace87

Wallace, R., R. Stockenburg, and R. Charette. *A Unified Methodology for Developing Systems.* New York: McGraw-Hill, 1987.

This book combines SADT with the Parnas/NRL design methodology [Gomaa89].

Ward85
Ward, P. T., and S. J. Mellor. *Structured Development for Real-Time Systems.* New York: Yourdon Press, 1985-1986.

Table of Contents

Volume 1
SECTION 1: INTRODUCTION
1 Historical Perspective
2 Characterization of Real-Time Systems
3 Real-Time Modeling Issues
4 Modeling Heuristics
5 The Modeling Process

SECTION 2: TOOLS
6 Modeling Transformations
7 Specifying Control Transformations
8 Specifying Data Transformations
9 Executing the Transformation Schema
10 Modeling Stored Data
11 Specifying Data
12 Organizing the Model
13 Integrating the Model Components

Index

Volume 2
1 Essential Modeling Heuristics
2 Defining System Context
3 Modeling External Events
4 Deriving the Behavioral Model
5 Completing the Essential Model—The Upper Levels
6 Completing the Essential Model—The Lower Levels
7 Essential Model Traceability

Appendix A: Cruise Control System
Appendix B: Bottle-Filling System
Appendix C: SILLY (Science and Industry Little Logic Yzer)
Appendix D: Defect Inspection System

The three volumes in this series are *Introduction and Tools*, *Essential Modeling Techniques*, and *Implementation Modeling Techniques*. The first two volumes are applicable to software requirements. Volume 3 covers software design.

Ward89
Ward, P. T. "Embedded Behavior Pattern Languages: A Contribution to a Taxonomy of CASE Languages." *J. Syst. and Software 9*, 2 (Feb. 1989), 109-128.

Abstract: *With the increasing availability of CASE tools, graphics-based software modeling languages have the potential to play a much more central role in the development process. Although some comparisons among these languages have been made, no systematic classification based on the underlying abstractions has been attempted. As a contribution to such a classification, a class of languages designated Embedded Behavior Pattern (EBP) languages is described and its members are compared and contrasted. The EBP languages include the Ward/Mellor and Boeing/Hatley Structured Analysis extensions, the Jackson System Development notation, and Harel's StateChart-Activity Chart notation. These notations are relevant to the building of specification models because they display clear one-to-one correspondences between elements of the model and elements of the application domain. These notations are also amenable to a style of model partitioning that is related to object-oriented development.*

This paper is a detailed comparison of the notations described in [Harel88a], [Hatley87], and [Ward85]. It will be particularly useful to instructors selecting real-time–oriented CASE tools to support requirements definition.

White87
White, S. *A Pragmatic Formal Method for Computer System Definition.* Ph.D. Th., Polytechnic Institute of New York, Brooklyn, N. Y., June 1987. Available from University Microfilms International, Ann Arbor, Michigan.

This thesis compares in more detail than any other publication the representational capabilities of several real-time requirements and design methods, including those described in [Alford85], [Ross85], [Heninger80], [Ward85], and [Harel88a]. White proposes a method that is claimed to combine the best characteristics of each of the evaluated methods. This thesis is recommended reading for researchers in requirements definition methods and anyone doing an in-depth comparison of real-time methods.

Yourdon89a
Yourdon, E. *Modern Structured Analysis.* Englewood Cliffs, N. J.: Yourdon Press, 1989.

Probably the most comprehensive and up-to-date book on the popular Structured Analysis method. Includes material on the real-time extensions to Structured Analysis and Entity-Relationship modeling. There are also two detailed case studies. If you need one book on Structured Analysis, this is probably the one to get.

Yourdon89b
Yourdon, E. *Structured Walkthroughs, 4th Ed.* Englewood Cliffs, N. J.: Yourdon Press, 1985. Also published by Prentice-Hall, 1989.

This book is the most comprehensive available on walkthroughs. The appendix "Guidelines for Analysis Walkthroughs" is particularly relevant.

UNLIMITED, UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS NONE
2a SECURITY CLASSIFICATION AUTHORITY N/A	3 DISTRIBUTION/AVAILABILITY OF REPORT APPROVED FOR PUBLIC RELEASE DISTRIBUTION UNLIMITED
2b DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) SEI-CM-19-1.2	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION SOFTWARE ENGINEERING INST. — 6b. OFFICE SYMBOL (If applicable) SEI	7a. NAME OF MONITORING ORGANIZATION SEI JOINT PROGRAM OFFICE
6c. ADDRESS (City, State and ZIP Code) CARNEGIE MELLON UNIVERSITY PITTSBURGH, PA 15213	7b. ADDRESS (City, State and ZIP Code) ESD/AVS HANSCOM AIR FORCE BASE HANSCOM, MA 01731
8a. NAME OF FUNDING/SPONSORING ORGANIZATION SEI JOINT PROGRAM OFFICE — 8b. OFFICE SYMBOL (If applicable) ESD/ AVS	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F1962890C0003
8c. ADDRESS (City, State and ZIP Code) CARNEGIE MELLON UNIVERSITY PITTSBURGH, PA 15213	10 SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO. 63752F · PROJECT NO. N/A · TASK NO. N/A · WORK UNIT NO. N/A
11. TITLE (Include Security Classification) Software Requirements	
PERSONAL AUTHOR(S) John W. Brackett, Boston University	
13a. TYPE OF REPORT FINAL — 13b. TIME COVERED FROM ______ TO ______	14. DATE OF REPORT (Yr., Mo., Day) January 1990 — 15. PAGE COUNT 27
16. SUPPLEMENTARY NOTATION	
17. COSATI CODES FIELD · GROUP · SUB. GR.	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) requirements requirements analysis requirements identification

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

This curriculum module is concerned with the definition of software requirements--the software engineering process of determining what is to be produced--and the products generated in that definition. The process involves all of the following: requirements identification; requirements analysis; requirements representation; requirements communication; and development of acceptance criteria and procedures. The outcome of requirements definition is a precursor of software design.

DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☒	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED, UNLIMITED DISTRIBUTION
22a. NAME OF RESPONSIBLE INDIVIDUAL JOHN S. HERMAN, Capt, USAF	22b. TELEPHONE NUMBER (Include Area Code) 412 268-7630 — 22c OFFICE SYMBOL SEI JPO