Managing Software Risks with Metrics and Measures

Robert A. Martin

24 October 2002

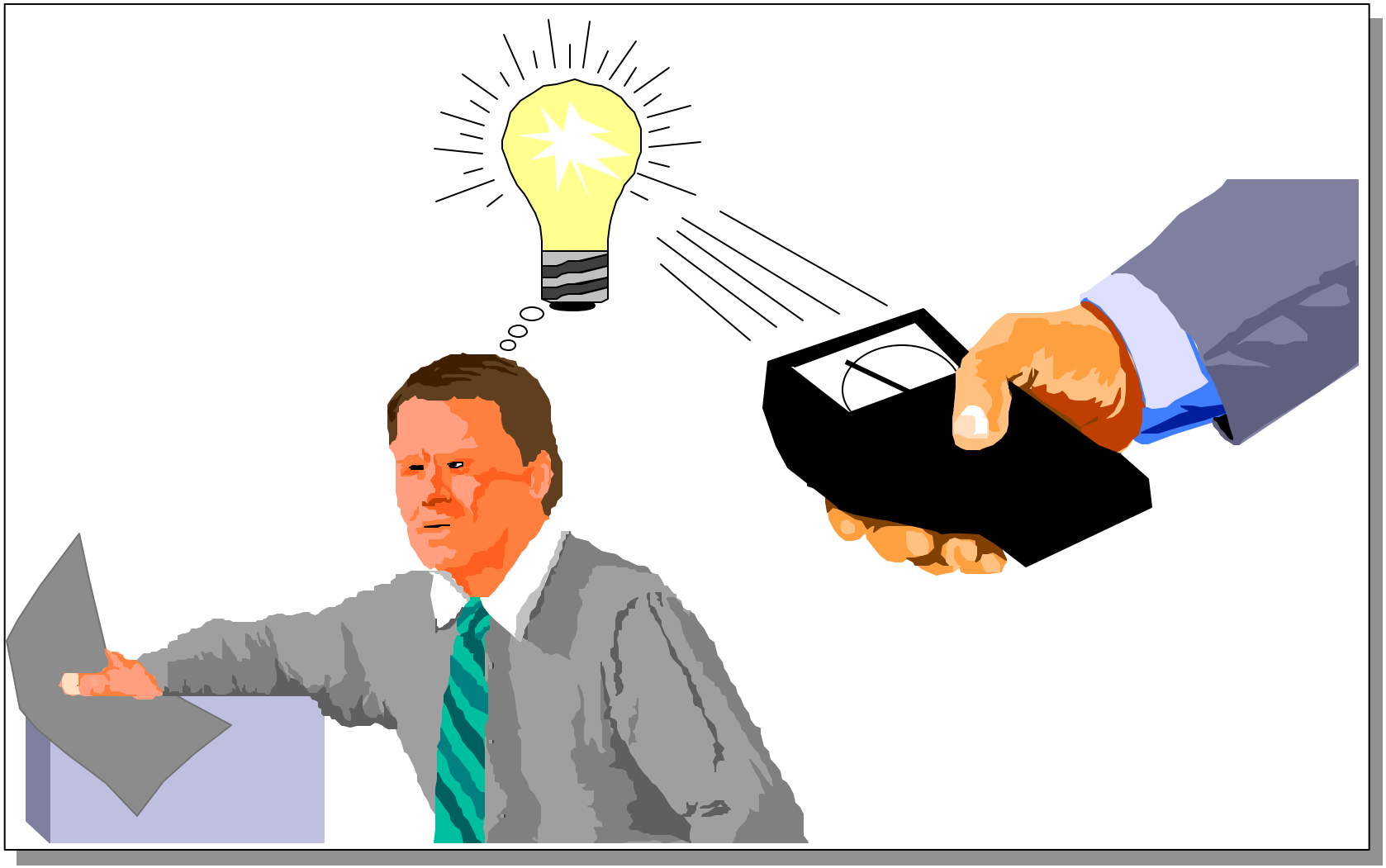
The views expressed in this presentation are those of the authors and do not necessarily reflect the policies or position of The MITRE Corporation.
Editorial graphics © 1994-2002 Martin and Morrison, used with permission.

MITRE

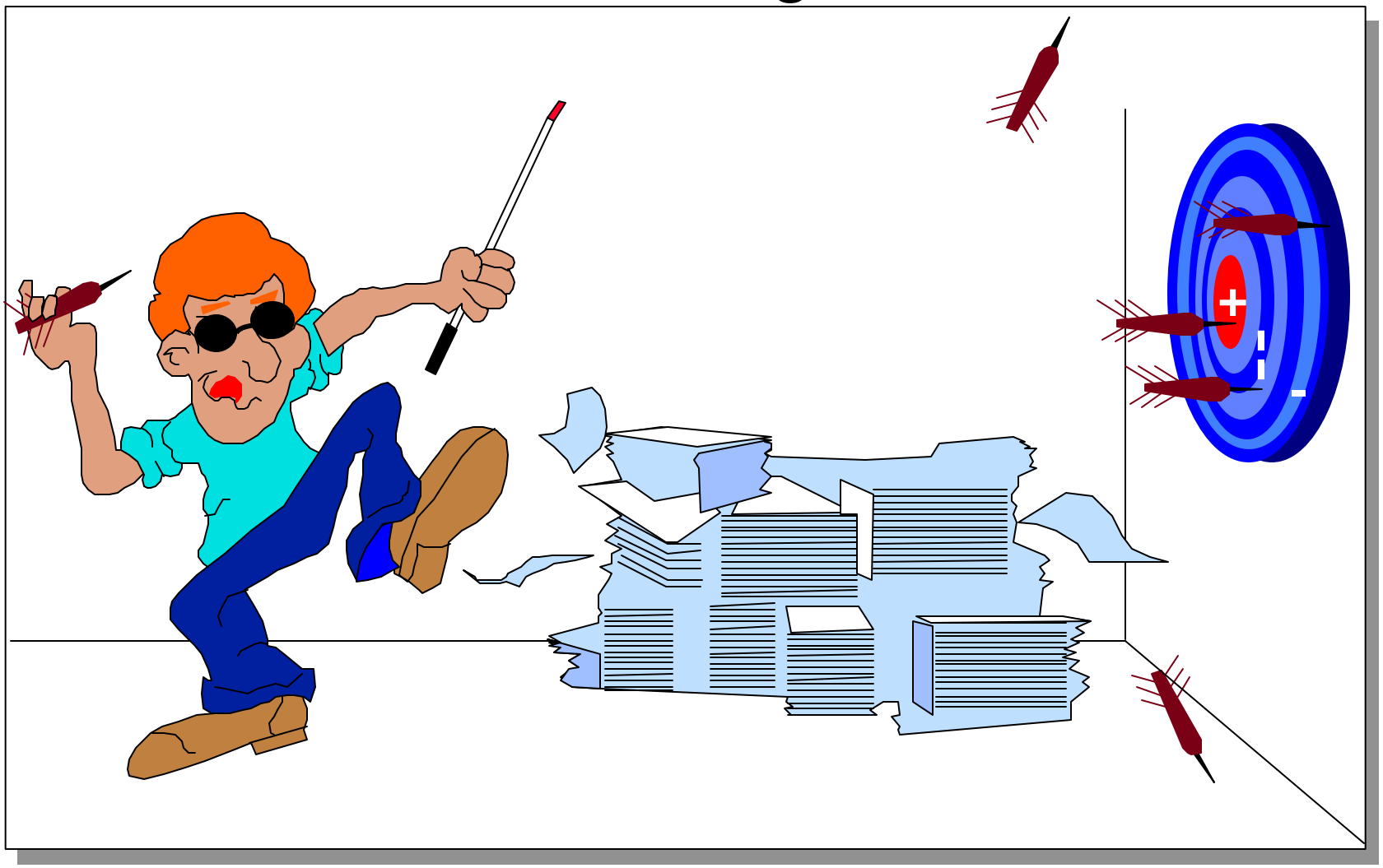
Discussion Outline

- 0 **Introduction**
  - **Managing S/W Quality Issues by Measuring Risks**
- 0 **Background**
  - **S/W Quality Risks**
  - **Metrics and Measures**
  - **Using Risk Assessments to Manage**
- 0 **Discussion**
  - **Components of a Risk Management Framework**
  - **Transferring to Industry and Academia**
  - **Including OO Specific Risks**
- 0 **Summary**

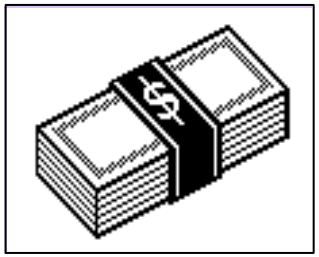
If, To Manage You Must Measure… How Do You Measure an Abstract Concept Like Software Risk?



One Method of Assessing Software Risks

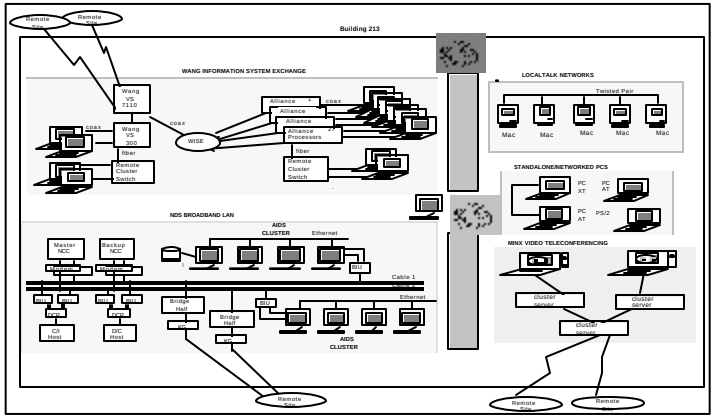


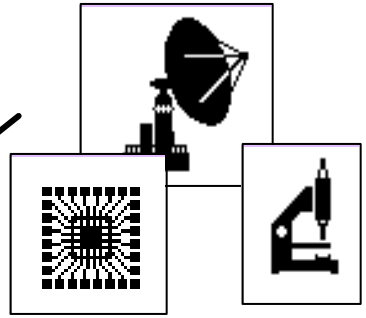
Why Are We Interested In S/W Risk Anyway?



Constrained Resources

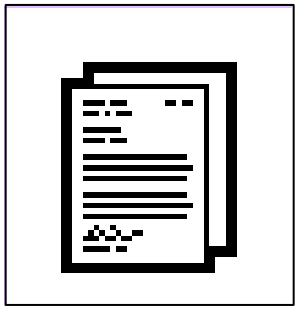
C^3I Systems





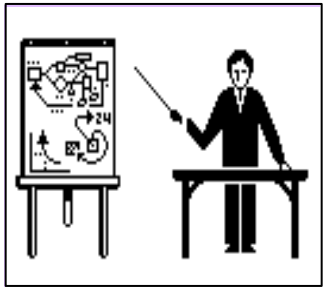
New Technology

New Requirements and Needs



change

Changing Operational Environment



Software Quality Teams must do more than remove errors!

MITRE

Looking Beyond Errors To Judge The Risks in Software

- Traditionally, most software organizations have focused on:
  - Managing the initial development schedule
  - Managing the development costs
  - Providing desired initial functionality to users
- Maintainability issues are frequently deferred until the product is fielded
- Why should a look at risks focus on the long-term perspective?
  - Software outlives hardware
  - Tightening budgets motivating code re-use efforts
  - Decisions made early in development may mean the difference between updating code and re-engineering it

Historically, eighty cents out of every dollar spent on software goes toward maintenance.

Defining Software Risk Issues

- What is the system I currently have really like?
  - Type & level of detail in the documentation
  - Understandability of the code and documentation
- What flexibility has been left to me by the original developers?
  - Hard coded assumptions, ties to proprietary features, or other limitations
  - Overall capacity for functional growth and change
- How tied am I to my current environment?
  - OS & COTS dependencies
  - Choice of language, tools, and/or language extensions

The real question is, what is the risk that the system will never work again after we make a change!

Introduction: MITRE's Software Risk Assessment Work

- Desired a flexible methodology for assessing s/w systems
  - Apply to any language, any platform, any architecture, …
- To be supported by a set of assessment technologies
- Must be an objective s/w-centric profile of the quality risks
- Should be a framework for assessing lifecycle quality issues
- Risk profiles would be usable in:
  - Selecting contractors based on quality of past efforts
  - Monitoring software quality during development
  - Identifying potential development process changes
  - Guiding future migration decisions
- Resultant framework, methods, and tools have been proven
  - > 100 Systems, ~ 51 million lines of code, >52 languages from a multitude of architectures (UNIX varieties, VMS, MVS, Windows, Macintosh)
  - Systems range from 4K to 6,254K lines of code -- average of 500K

Discussion Outline

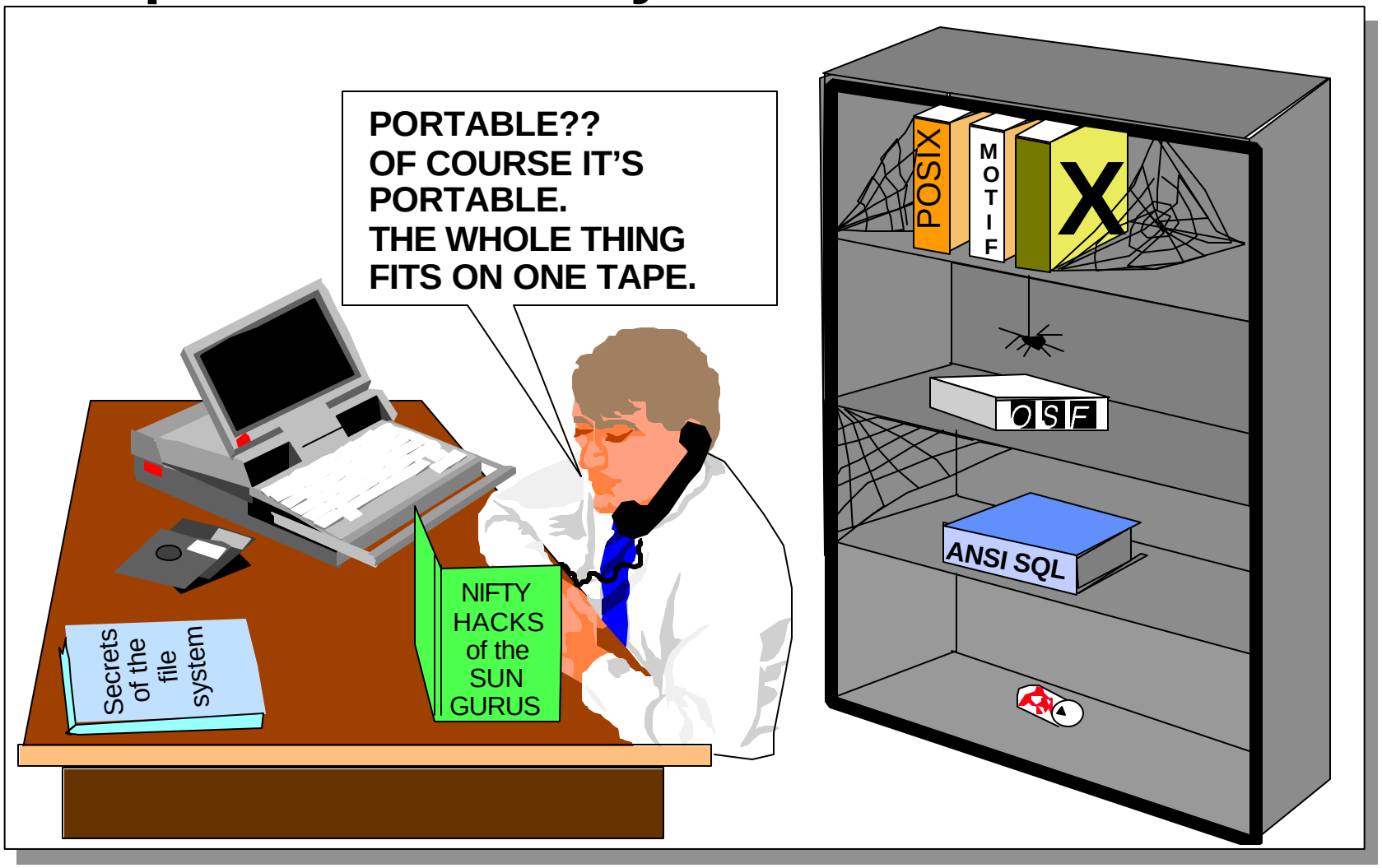
- 0 **Introduction**
  - **Managing S/W Quality Issues by Measuring Risks**
- 0 **Background**
  - **S/W Quality Risks**
  - **Metrics and Measures**
  - **Using Risk Assessments to Manage**
- 0 **Discussion**
  - **Components of a Risk Management Framework**
  - **Transferring to Industry and Academia**
  - **Including OO Specific Risks**
- 0 **Summary**

Software Risks Impact on Quality Are Not Well Defined

- Most will agree they want their systems to be reliable, maintainable, evolvable, portable, open, etc.
- Most people can't agree on what, specifically, reliable, maintainable, evolvable, portable, open, etc. actually mean or how to measure such qualities for an arbitrary body of code
- Commercial software tools and metrics provide insights into implementations but typically do not provide any sense of higher context for lifecycle issues

Our definition: A quality system minimizes the risks to the system

Developers Can Provide Plenty of Complications to a System's Software Risks



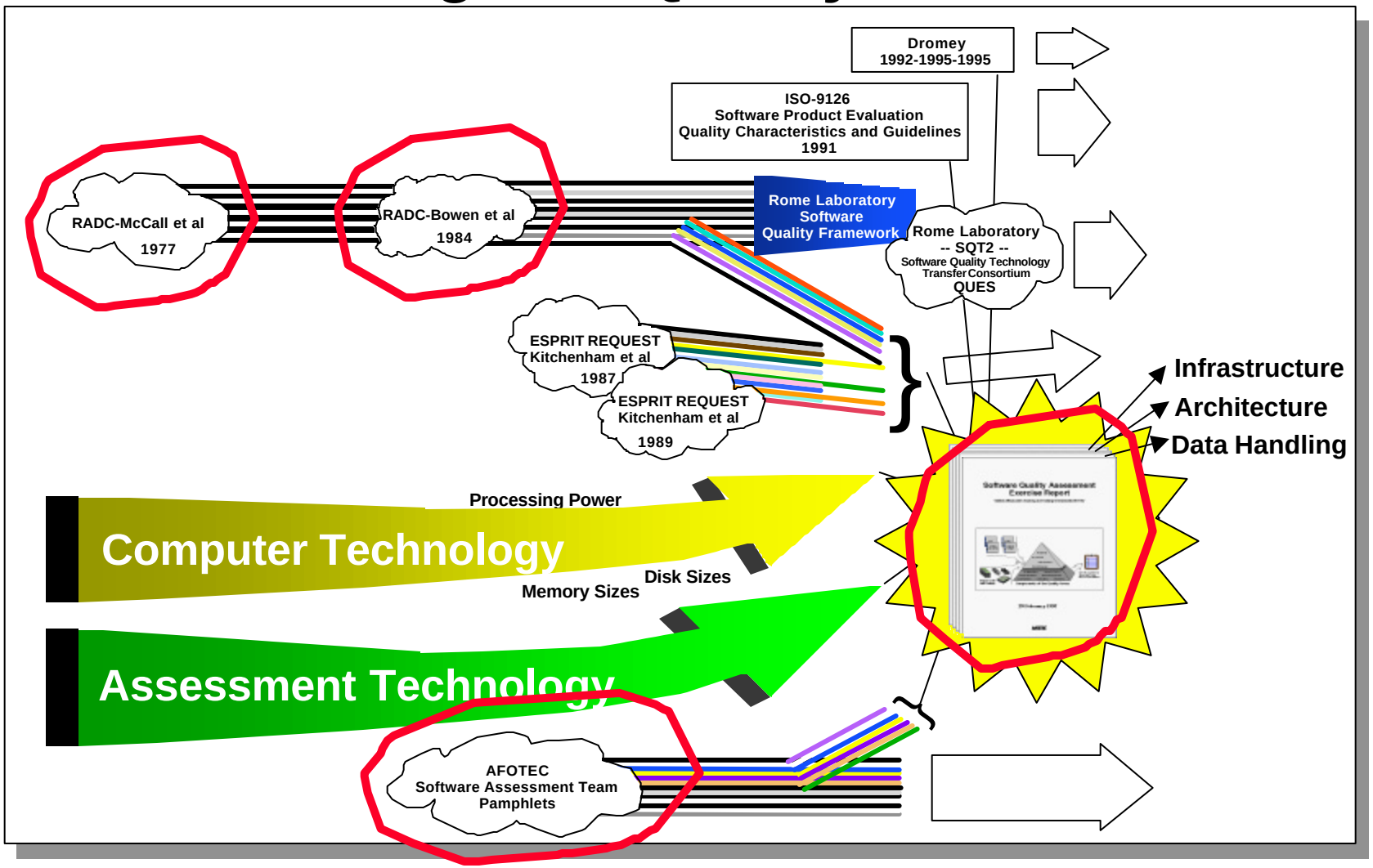
Establishing a Framework for Measuring Risks

- **Many areas can help minimize a system's risks**
  - **Some are well studied and have full fledged disciplines, technologies, and examination methodologies in place**
  - **Specifically: requirements traceability, functional completeness, and system testability are well established areas of study**
- **The other life-cycle risk areas have received less attention but have enormous potential for reducing the levels and types of risk in the systems fielded**
- **Much to draw from:**
  Rome Air Development Center work and others
  - **McCall et al. in 1977**
  - **Bowen et al. in 1984**
  - **Kitchenham et al.'s ESPRIT REQUEST project, 1987 & 1989…**

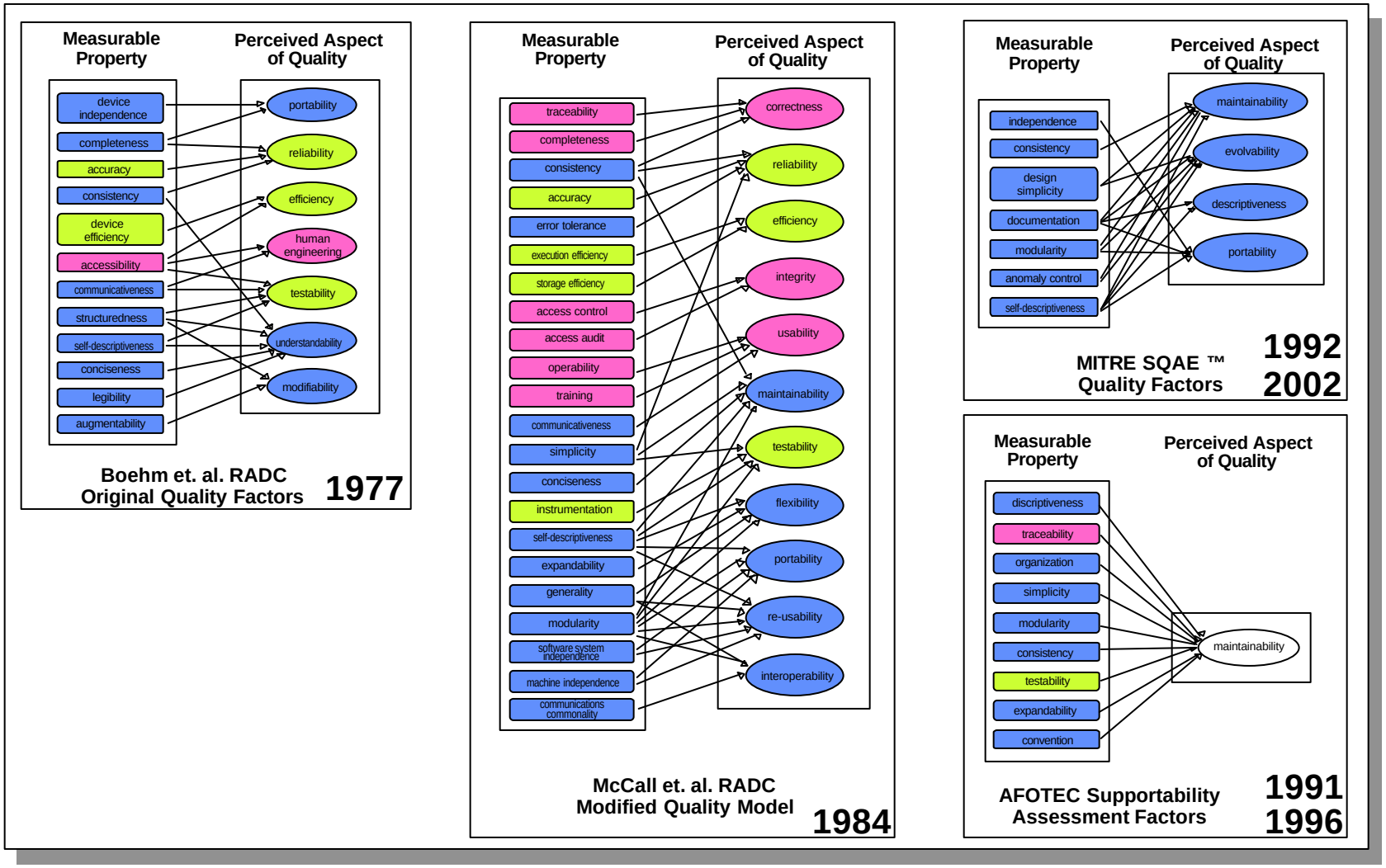
Attributes Of A Useful S/W Quality Risk Assessment Methodology

- **The assessment should be:**
  - **repeatable (independent of the assessor(s))**
  - **independent of language, architecture, platform**
  - **not dependent on presence of “all” code**
  - **provide detailed insight into the software risks**
  - **software centric**
  - **based on artifacts only**
  - **“cheap” to perform**
  - **examine all artifacts of the system**
    - **source code (including scripts, data, …)**
    - **supporting documentation (both internal and external to the code) and standards**
  - **leverage automation where-ever possible**

There Are Several Frameworks for Evaluating and Monitoring S/W Quality Risks



S/W Quality Risk Frameworks: Scope and Focus Can Differ



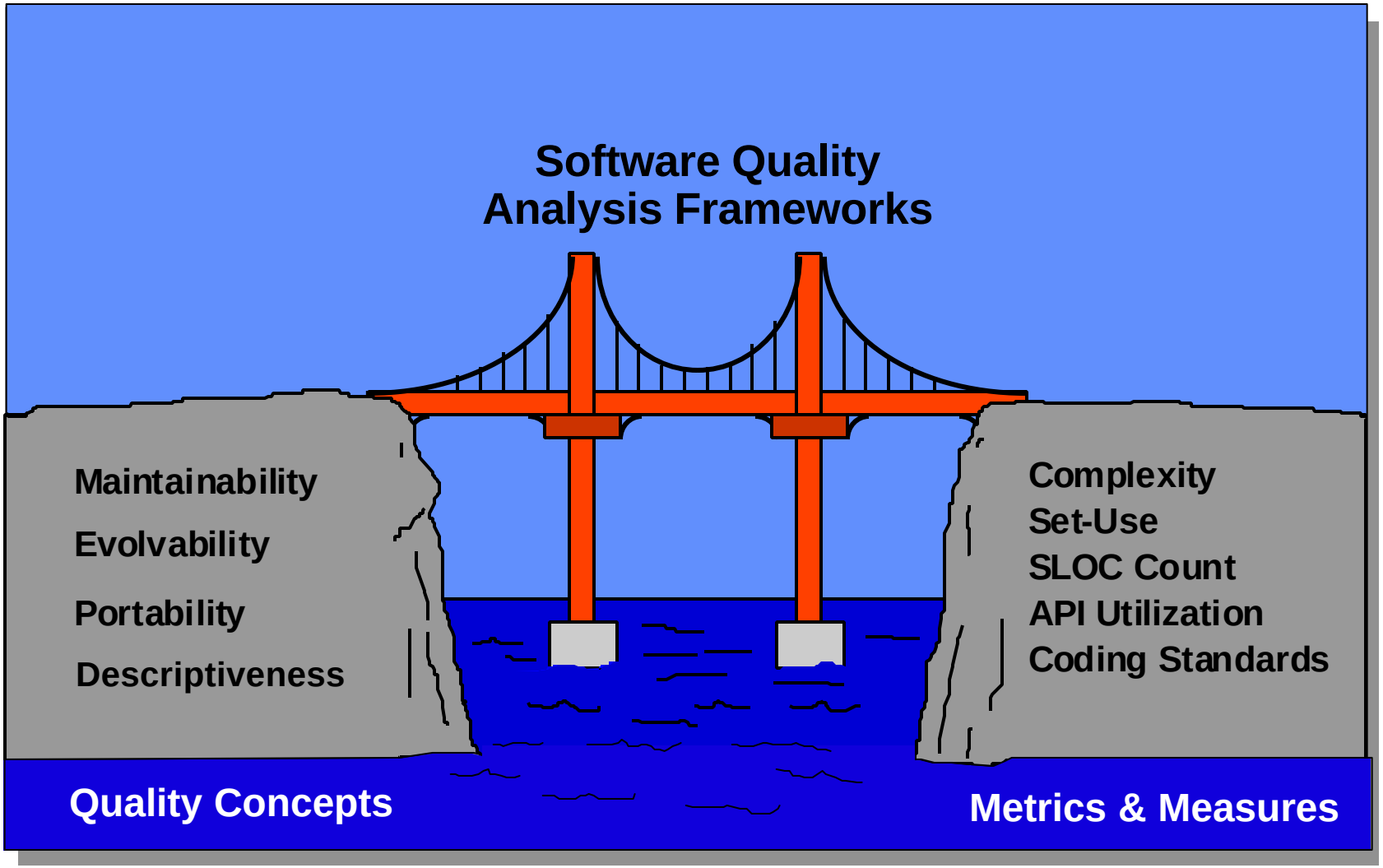
- software attributes
- requirements issues
- testing issues

MITRE

Discussion Outline

- 0 **Introduction**
  - **Managing S/W Quality Issues by Measuring Risks**
- 0 **Background**
  - **S/W Quality Risks**
  - **Metrics and Measures**
  - **Using Risk Assessments to Manage**
- 0 **Discussion**
  - **Components of a Risk Management Framework**
  - **Transferring to Industry and Academia**
  - **Including OO Specific Risks**
- 0 **Summary**

Bridging the Gap between The Measurable and Unmeasurable



Guiding Principles: Breadth, Depth, and Repeatability

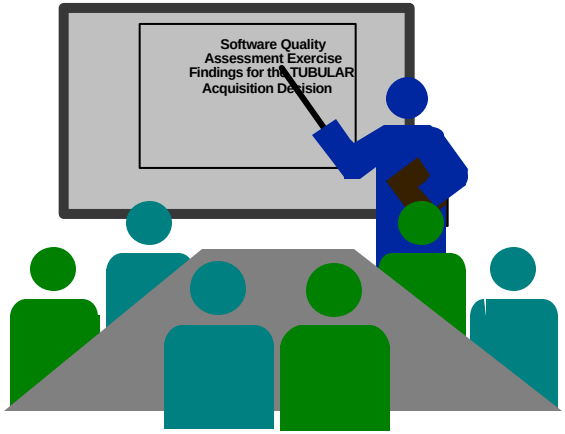
- The evaluation of each quality issue should have a specific scope and context as well as a defined scoring criteria
- Define context for ratings (ideal, good, marginal, and fail)
  - limiting choices increases repeatability
- Use a mixture of:
  - Hard metrics (cyclomatic complexity, flow complexity, …)
  - Objective measures (type of information available, existence
  - Subjective measures (use of white space, usefulness
- The Metrics and Objective Measures attributes can have a scope of all of the code of the system
- The Measures which require cognitive reasoning need to be scoped more narrowly (7/7/7 per language)
- Provide a software tools framework to guide and assist evaluators & provide context and control of the process

Discussion Outline

- 0 **Introduction**
  - **Managing S/W Quality Issues by Measuring Risks**
- 0 **Background**
  - **S/W Quality Risks**
  - **Metrics and Measures**
  - **Using Risk Assessments to Manage**
- 0 **Discussion**
  - **Components of a Risk Management Framework**
  - **Transferring to Industry and Academia**
  - **Including OO Specific Risks**
- 0 **Summary**

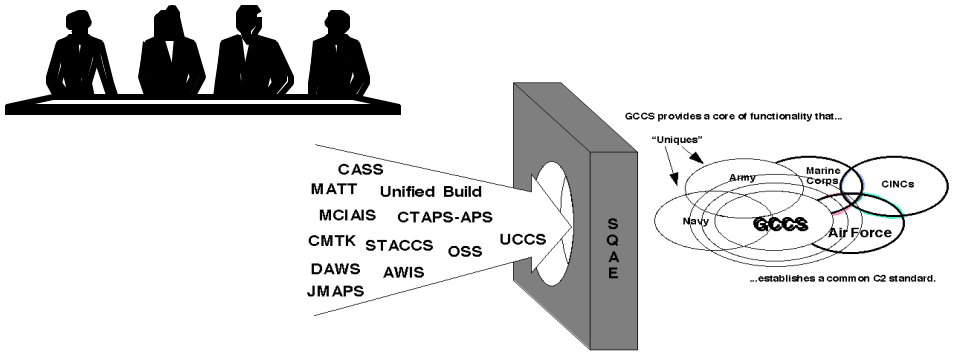
Having An Understanding Software Quality Risks Can Be Used In…

The Selection of Contractors



Reviews of S/W Releases for a Project Office

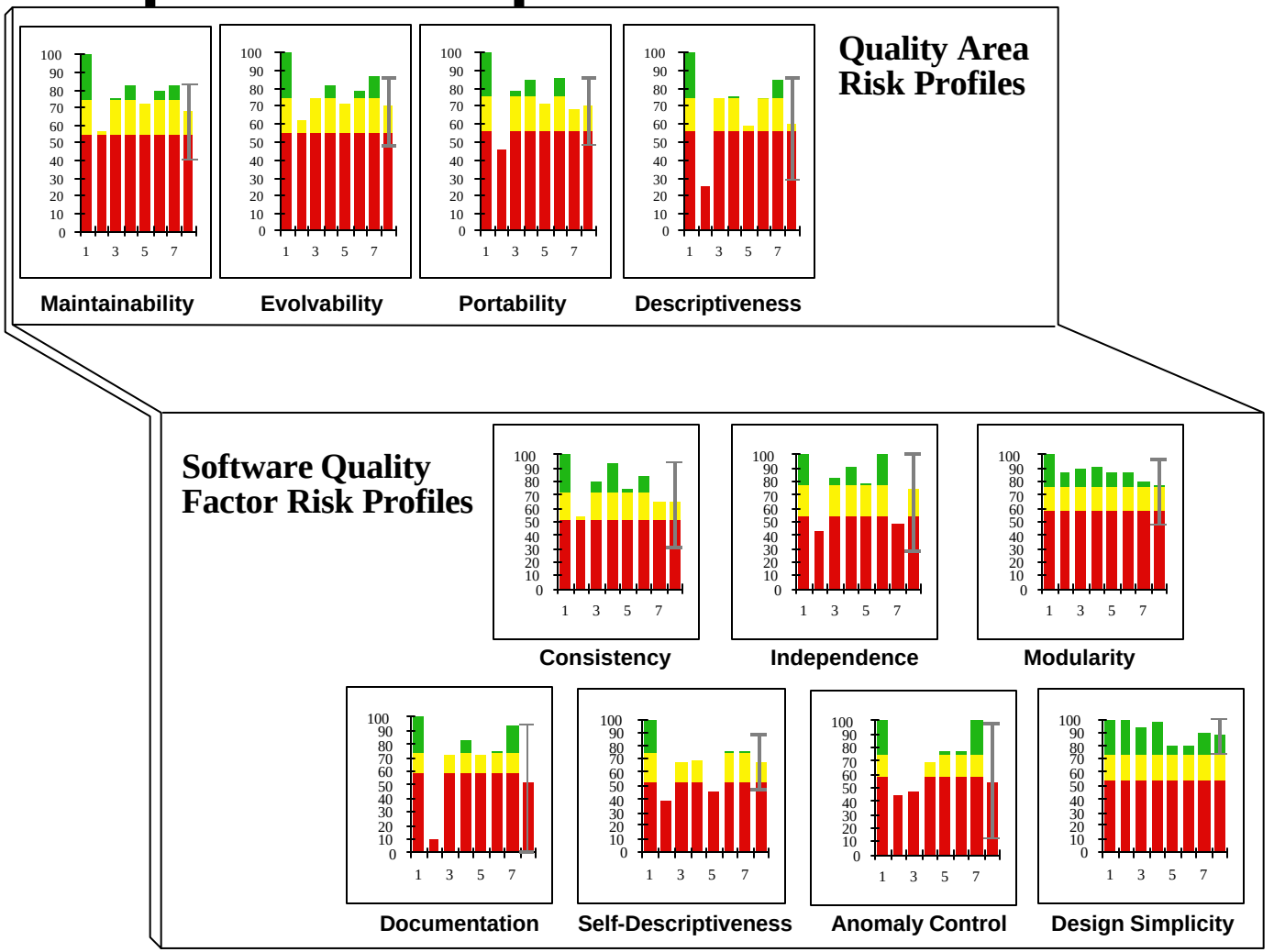
Selection of Migration Systems



Software Quality Assessment Uses

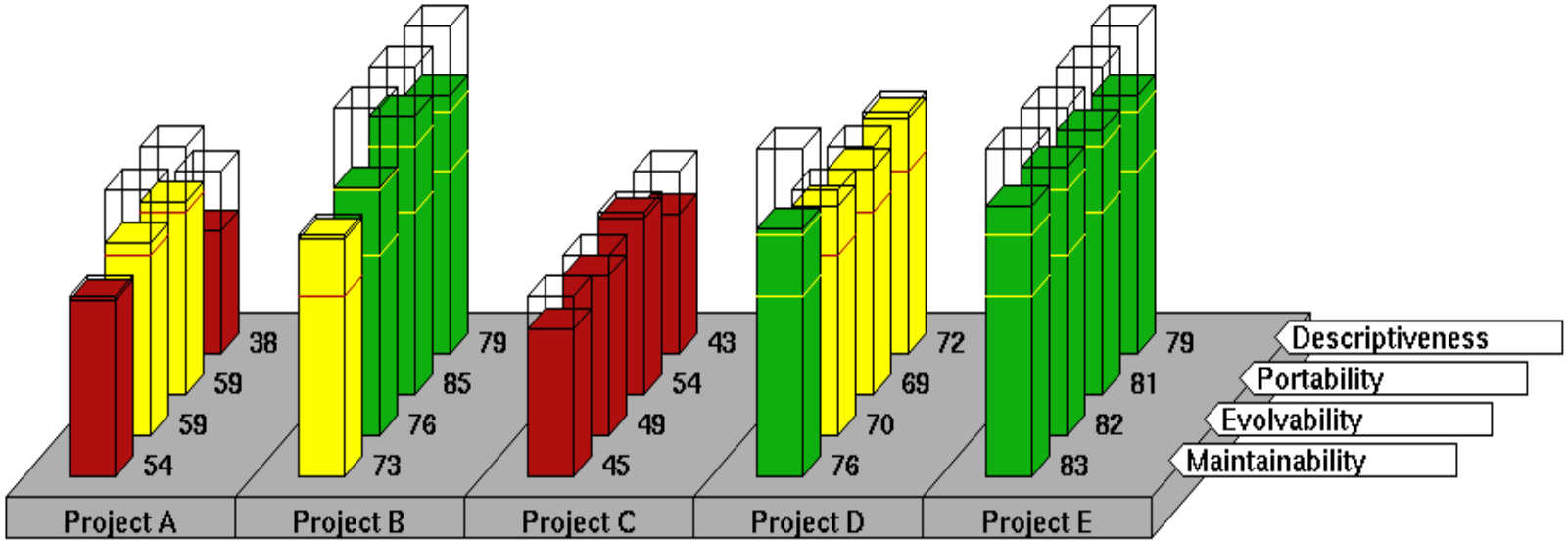
- **Understanding the Software's quality can:**
  - **Allow for evaluation of a contractor based on quality of past products**
  - **Allow for in-progress corrections to a development effort**
  - **Guide future migration decisions**
  - **Provide for the rapid identification of the sources of risk**
    - **in understandable & actionable terms for mgmt**
    - **in fine detail for the technologists**
  - **Provide a broad review of the software lifecycle risks associated with multi-component systems**
  - **Allow risk comparisons for systems independent of language, platform, architecture, …**
  - **Guide the build, buy, or re-use decisions**

Sample Assessment Results for Multiple Developers



Key
Data Series 1 = Reference
Data Series 2 = Prj 1 Vndr A
Data Series 3 = Prj 2 Vndr A
Data Series 4 = Prj 1 Vndr B
Data Series 5 = Prj 2 Vndr B
Data Series 6 = Prj 1 Vndr C
Data Series 7 = Prj 2 Vndr C
Data Series 8 = Mean & Range of 40 Systems

Examples of Software Quality Risk Profiles (3D)



This Chart Contains Representative Assessment Results

Examples of Feedback

Application's Primary Strengths: Integrator Perspective

- Isolation of dependencies
  - Effort has been made to segregate code so that actual processing algorithms are buffered from platform and COTS dependencies.
  - This buffering lowers the system's sensitivity to changes in its operating environment.
  - Should the platform change significantly (N
    Database, etc) code rewrites and unit tests
    restricted to distinct areas rather than rippl
    system.

Application's Primary Weaknesses: Integrator Perspective

- Descriptiveness
  - The provided documentation addresses aspects of the system only at the highest level and does not detail essential low level information:
    - System dependencies
    - Knowledge domains required for maintenance
    - Input data tolerance and valid range of value definitions
    - Specific data flow descriptions
    - Policies for error handling
  - The code itself is poorly documented internally and makes frequent use of programming constructs which hinder readability and traceability

MITRE

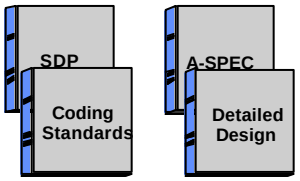
Discussion Outline

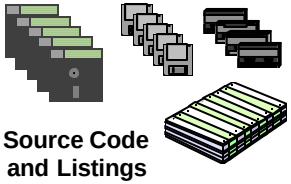
- 0 **Introduction**
  - **Managing S/W Quality Issues by Measuring Risks**
- 0 **Background**
  - **S/W Quality Risks**
  - **Metrics and Measures**
  - **Using Risk Assessments to Manage**
- 0 **Discussion**
  - **Components of a Risk Management Framework**
  - **Transferring to Industry and Academia**
  - **Including OO Specific Risks**
- 0 **Summary**

Methodology and Process: The Software Quality Assessment Exercise (SQAE™)

Approach: Assess the level of risk in the Quality Areas by examining the underlying attributes of the code and documentation.

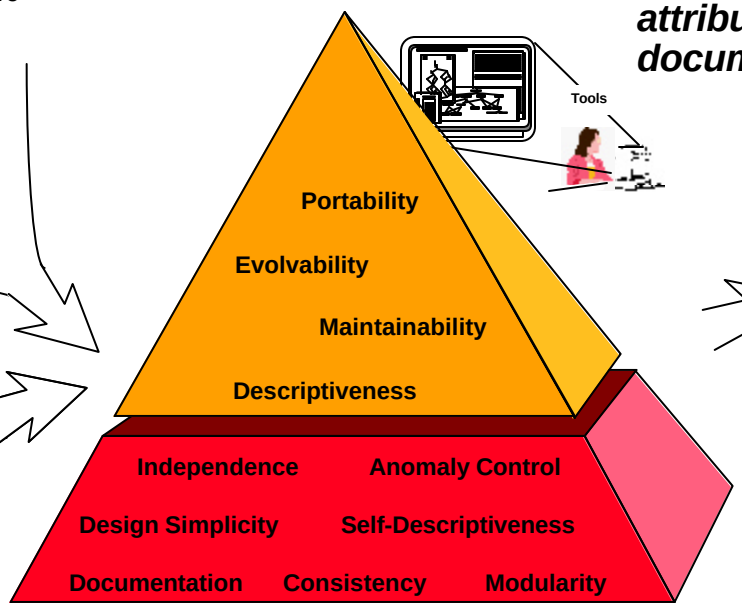
Coding Standards, Software Development Plan, Design Documents, etc.





Source Code and Listings

Specific Questions about the Code and Documentation



Components of the Software Quality Areas

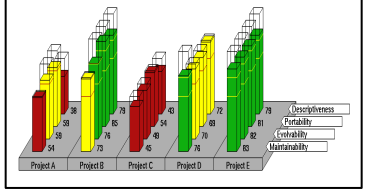
Quality Risk Profile: Risk Mitigators

- Naming conventions used for modules and variables helps understand the code's functionality
- Good use of white space and indention
- Modules are easily viewed at once (< 100 LOC)
- Good functional documentation with high-level design
- Good design documentation, showing data and control flows.
- Good developer documentation for supported APIs
- Good top-down hierarchical structure to code
- Modules use straightforward algorithms in a linear fashion
- System dependencies are to readily available COTS software
- Code is of low complexity.
- Logic flow through individual procedures is easy to follow.
- Disciplined coding standards followed by the programmers.
- Considerable effort made to use POSIX calls throughout.
- System dependencies are isolated and all dependencies on the platform or COTS are encapsulated.

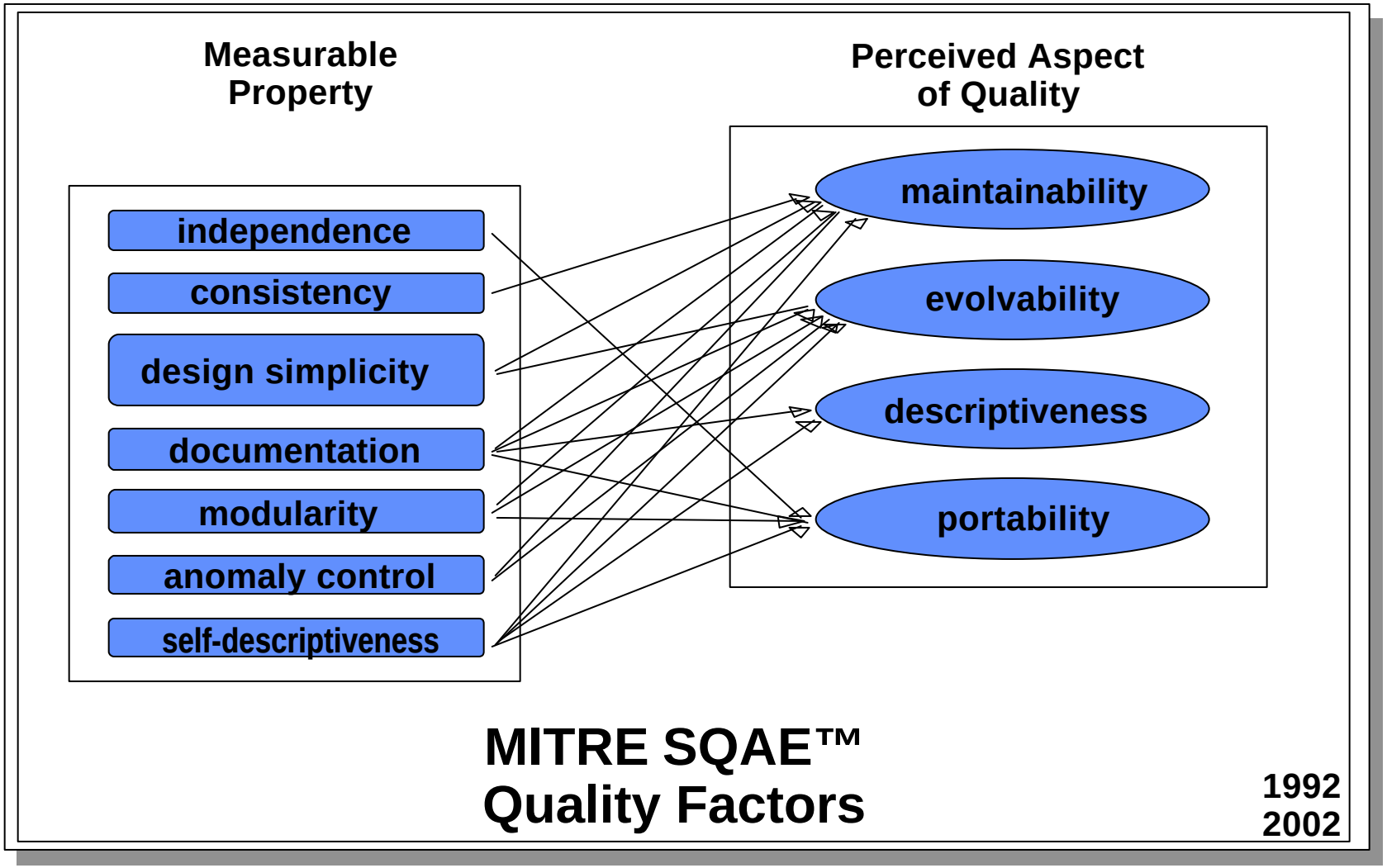
Quality Risk Profile: Risk Drivers

- Level of isolation and encapsulation of dependencies on platform and COTS packages varies between programmers.
- Use of environmental variables is undocumented and inconsistently done.
- Lack of written standards for naming conventions, error handling, data definitions, etc.
- Lack of standards for naming conventions, error handling data definitions, I/O, etc.
- Design documentation is poorly organized, incomplete and at a very high level.
- No low-level design information or functional allocation of software in documentation
- Machine generated code documentation is inconsistent with the developed code documentation.
- Machine generated code is undocumented.
- Hard-coded absolute filenames/pathnames lack uniqueness.
- Hard-coded absolute filenames/pathnames used
- UNIX commands hardcoded in the code
- Hard-coded variables used when symbolic constants should have been used
- There are some machine dependent data representations
- Code is not ANSI standard
- Variables used for other than their declared purpose
- No low-level control and task flows in documentation
- No prologs for the majority of the modules
- Poor quality of design documentation
- Excessive use of global variables
- Input error checking is not consistently applied
- System dependent on a proprietary language for some functions related to integration with COTS.
- Lack of consistency in the code between programmers.
- No isolation or encapsulation of dependencies on platforms or COTS.
- System tied to a proprietary language for procedural processing and data access.
- System is dependent on a proprietary run-time environment.
- Fourteen violations of one of the few company coding standards.
- Two percent of the code modules are overly large, more than 100 LOC.

Software Quality Risk Profiles



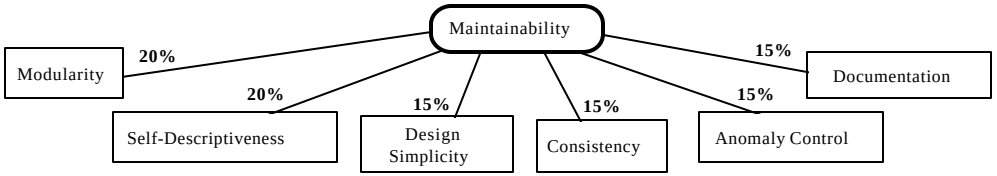
Test & Requirements Issues are Addressed - So We Focused on S/W Attribute Issues

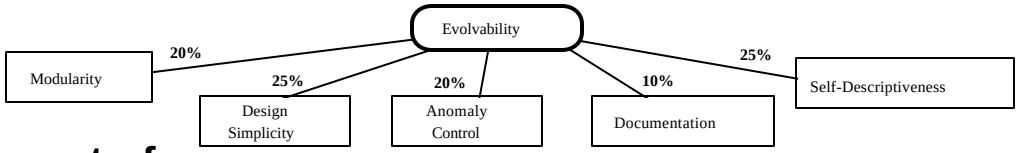


- software attributes

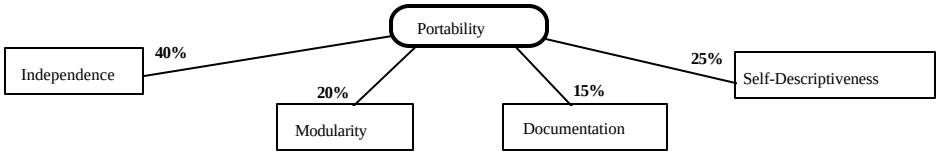
Details: SQAE™ Areas and Factors

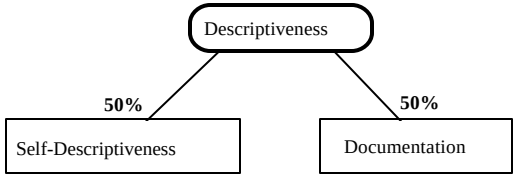
- Assess software against a defined set of quality areas:
  - Portability
  - Evolvability
  - Maintainability
  - Descriptiveness





- Quality areas are based on a set of seven components:
  - Consistency (15 attributes)
  - Independence (8 attributes)
  - Modularity (10 attributes)
  - Documentation (16 attributes)
  - Self Descriptiveness (11 attributes)
  - Anomaly Control (5 attributes)
  - Design Simplicity (11 attributes)





Details of the SQAE™ Quality Component Assessment Questions

Independence Independence comprises two broad groups; software system independence and machine independence. Here the issue is to not tie the system to any specific host environment which would make it difficult or impossible to migrate, evolve, or enhance the system.

		Proportion of Factor Value
Software System Independence		
2.1	- Does the software avoid all usage of specific pathnames/filenames?	(100)
2.2	- Is the software free of machine, OS and vendor specific extensions?	(200)
2.3	- Are system dependent functions, etc., in stand-alone modules (not embedded in the code)?	(200)
2.4	- Are the languages and interface libraries selected standardized and portable? (i.e., ANSI…)	(200)
2.5	- Does the software avoid the need for any unique compilation in order to run (e.g., a custom post processor to "tweak" the code to run on machine X)?	(100)
2.6	- Is the generated code (i.e., GUI Builders) able to run without a specific support runtime component?	(100)
Machine Independence		
2.7	- Is the data representation machine independent?	(200)
2.8	- Are the commercial software components available on other platforms in the same level of functionality?	(200)

Modularity Modularity consists of several facets which each support the concepts of organized separation of functions and minimizes un-noticed couplings between portions of the system.

		Proportion of Factor Value
3.1	- Is the structure of the design hierarchical in a top-down design within tasking threads?	(200)
3.2	- Do the functional groupings of units avoid calling units outside their functional area?	(150)
3.3	- Are machine dependent and I/O functions isolated and encapsulated?	(150)
3.4	- Are interpreted code bodies (shell scripts and 4GL scripts) protected from accidental or deliberate modification?	(150)
3.5	- Do all functional procedures represent one function (one-to-one function mapping)?	(150)
3.6	- Are all commercial software interfaces & APIs, other than GUI Builders, isolated and encapsulated?	(200)
3.7	- Have symbolic constants been used in place of explicit ones?	(150)
3.8	- Are symbolic constants defined in an isolated and centralized area?	(100)
3.9	- Are all variables used exclusively for their declared purposes?	(150)
3.10	- Has the code been structured to minimize coupling to global variables?	(100)

Details of the SQAE™ Framework

Exercise A The first exercise area concentrates on those activities that can be accomplished by examining the two largest functional areas of the code. The activities in this exercise are listed below.

1.10	Are the naming conventions consistent for functional groupings?	Examine the scheduling modules and one other large functional grouping and cross reference between them. Rating will be either Ideal, Good, Marginal, or Failing. If at least one of the programmers is either consistent or uses distinguishable naming conventions (marginal), if he/she uses both (good), if all programmers do both (ideal).
2.2	Is the software free of machine, OS and vendor specific extensions?	Examine two large functional groupings of code and cross reference between them and system libraries and known vendor extensions. Rating will be either Ideal, Good, Marginal, or Failing. Score ideal if no instances occur, good if such assumptions affect less than 10% of the packages, marginal for less than 50%, else failing.
2.3	Are system dependent functions, etc., in stand-alone modules (not embedded in the code)?	Examine all known instantiations OS and vendor specific dependencies for encapsulation/isolation. Rating will be between 1 and 0, where 1 is the higher rating. 1 - (number of embedded dependencies/total number of dependencies)

⋮ ⋮ ⋮

SQAE™ Finding Examples: Mitigators, Drivers, & Other Observations

Risk Mitigators

- Naming conventions used for modules and variables helps understand the code's functionality.
- Good use of white space and indention.
- Modules are easily viewed at once (< 100 LOC)
- Good functional documentation with high-level design.
- Good design documentation, showing data and control flows.
- Good developer documentation for supported APIs.
- Good top-down hierarchical structure to code.
- Modules use straightforward algorithms in a linear fashion.
- System dependencies are to readily available COTS software.
- Code is of low complexity.
- Logic flow through individual procedures is easy to follow.
- Disciplined coding standards followed by the programmers.
- Considerable effort made to use POSIX calls throughout.
- System dependencies platform or COTS are encapsulated.

Other Observations

- No documented method for other languages to call services
- "Man pages" are out of date for some APIs
- Number of modules may be excessive
- COTS screen description files use standard X-Windows resource file formats
- Proprietary language does not support data typing
- In the vendor's proprietary language, variables are never explicitly declared (A typo will create a variable)
- SQL is only used for ~10% of the code that accesses the database
  - The rest uses the proprietary DBMS calls
- Complete source code for gnu Perl was included as part of deliverable subsystem source code

Risk Drivers

- Level of isolation and encapsulation of dependencies on platform and COTS packages varies between programmers
- Use of environmental variables is undocumented and inconsistently done
- Lack of written standards for naming conventions, error handling, data definitions, etc
- Lack of standards for naming conventions, error handling data definitions, I/O, etc
- Design documentation is poorly organized, incomplete, and at a very high level
- No low-level design information or functional allocation of software in documentation
- Machine generated code documentation is inconsistent with the developed code documentation
- Machine generated code is undocumented
- Procedure and file names depend on path for uniqueness
- Hard coded absolute filenames/paths used
- UNIX commands hardcoded in the code
- Hard coded variables used when symbolic constants should have been used
- There are some machine dependent data representations
- Code is not ANSI standard
- Variables used for other than their declared purpose
- No low-level control and task flows in documentation
- No prologs for the majority of the modules
- Inadequate indexing of documentation
- Excessive use of global variables
- Input error checking is not consistently applied
- System dependent on a proprietary language for some functions related to integration with COTS
- Lack of consistency in the code between programmers
- No isolation or encapsulation of dependencies on platform or COTS
- System tied to a proprietary language for procedural processing and data access
- System is dependent on a proprietary run-time environment
- Fourteen violations of one of the few company coding standards
- Two percent of the code modules are overly large, more than 100 LOC

This Chart Contains Representative Assessment Results

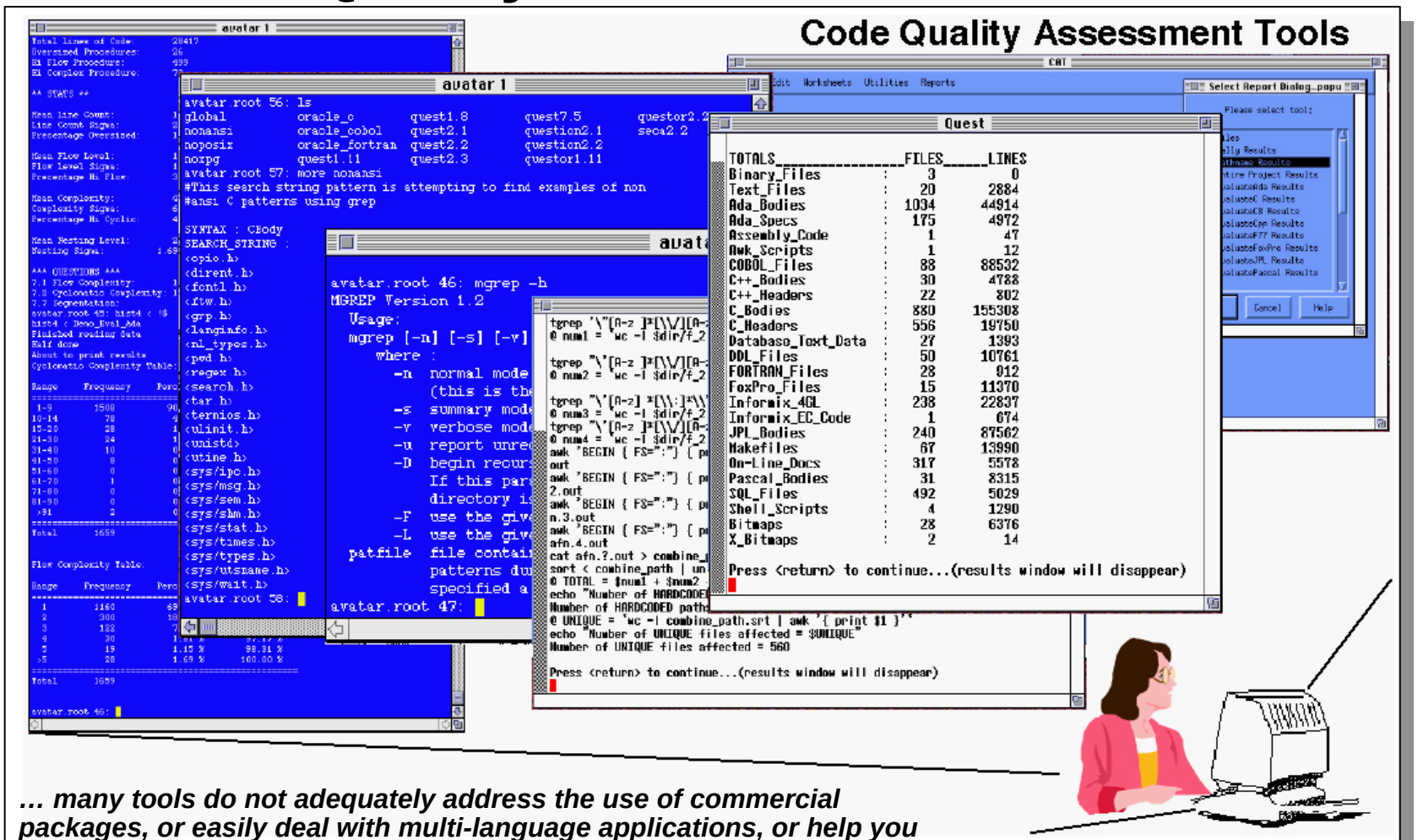
SQAE™ Foundation

	Project A	Project B	•••	Project CZ	Total of Projects
Source Code	112,000 LOC	558,000 LOC	•••	58,000 LOC	51,173,315 LOC
	Ada, C, Shell, TAE+, SQL, X, MOTIF, Stored Procedures	C, Shell, X, MOTIF	•••	Ada, C, ELF, ezX, SQL, X, MOTIF	Ada, C, FORTRAN, COBOL, shell, TAE+, SQL, X, MOTIF, UIL, Stored Procedures, GEL, ELF, ezX, ...
Written Material	Top Level Design Doc SDD	SDD SDP	•••	Top Level Design Doc SPS SDD SDP	Top Level Design Doc SPS SDD SDP Case Tools Repositories
Reference Material	Product Literature Reference Manual Users Manual	Design and Code Stnds	•••	Product Literature Reference Manual Users Manual Design and Code Stnds	Product Literature Reference Manual Users Manual Design and Code Stnds

COTS Manuals & Articles

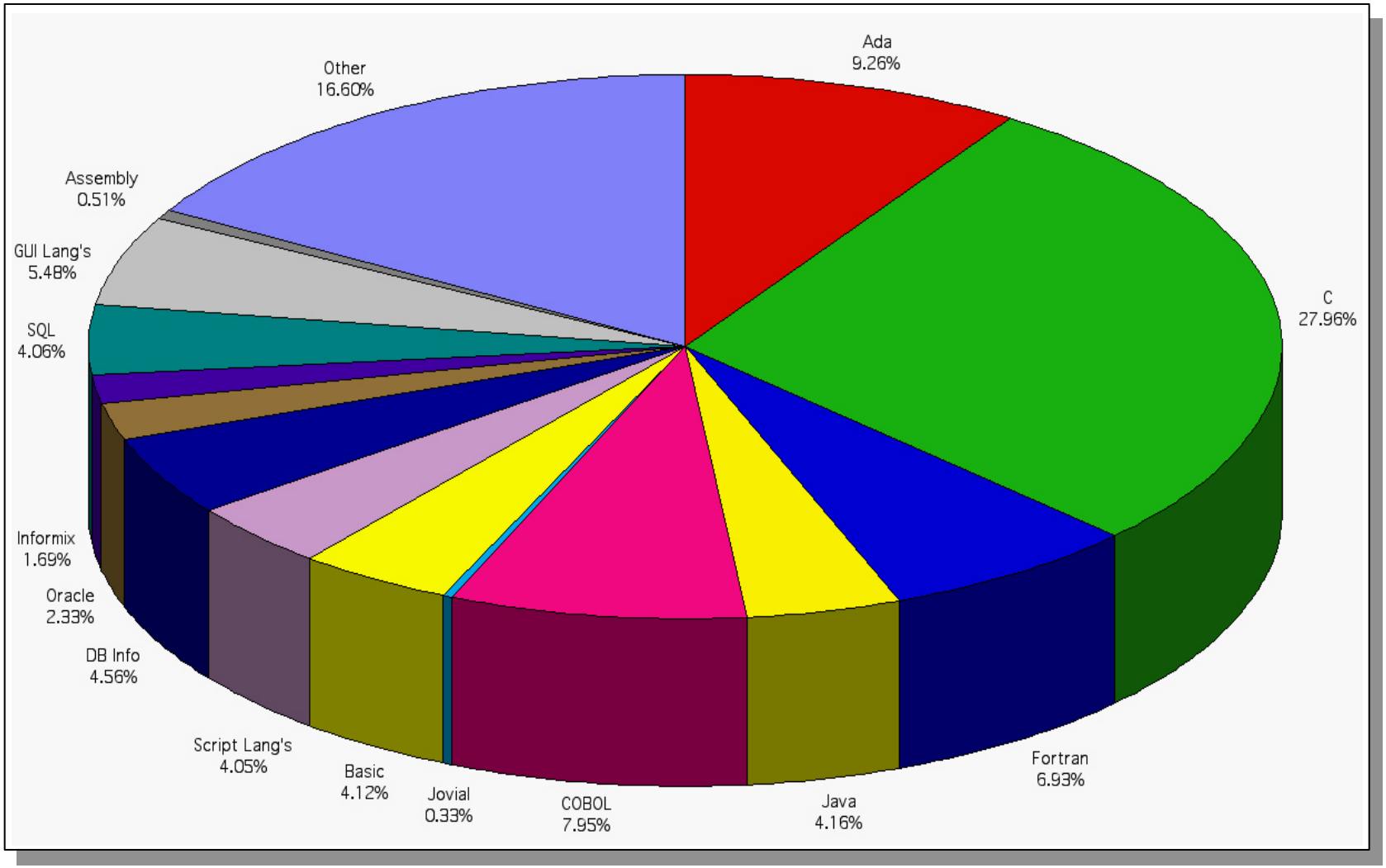
This Chart Contains Representative Assessment Results

Examples of Tools Used in Assessing Software Quality Risks



… many tools do not adequately address the use of commercial packages, or easily deal with multi-language applications, or help you correctly interpret their metrics.

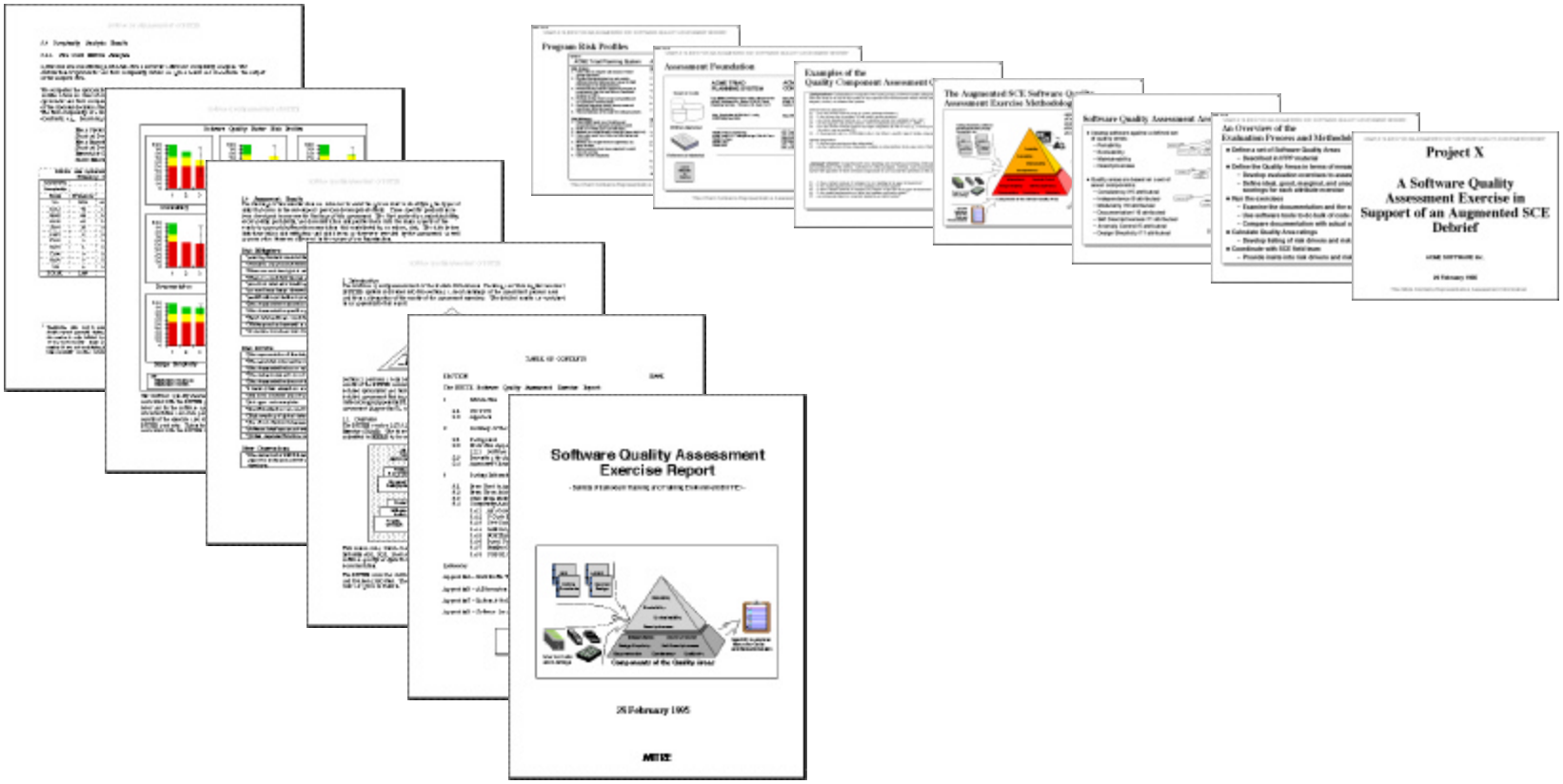
Languages from Product Assessments By Percentage of Assessed LOC



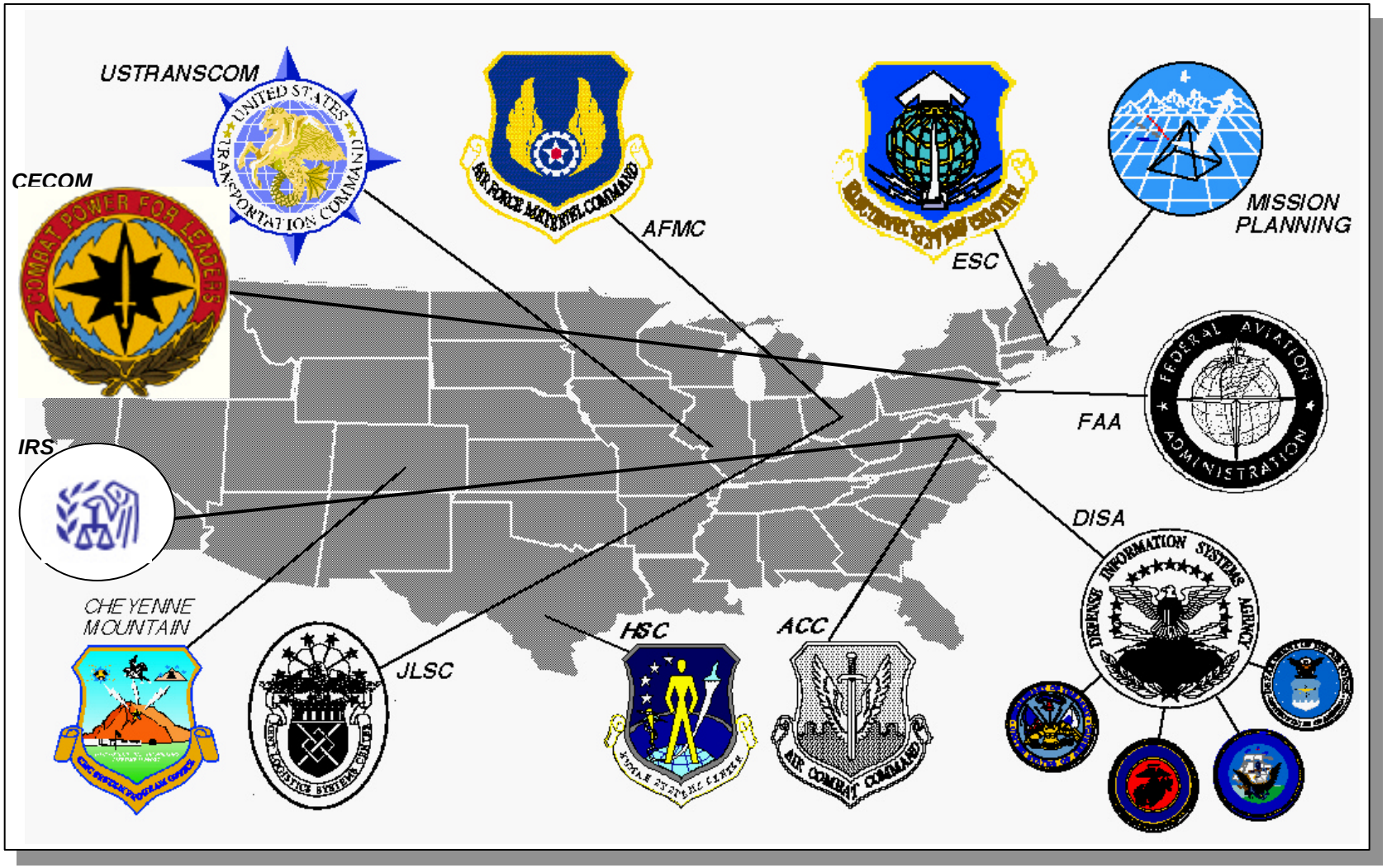
100 Systems with approximately 51 Million Lines of Code Assessed

Reporting the Results of a Software Quality Risk Assessment Exercise

Both a Software Quality Risk Assessment Report and (if desired) a Briefing version are available



SQAE™ Experience-Base



Discussion Outline

- 0 **Introduction**
  - **Managing S/W Quality Issues by Measuring Risks**
- 0 **Background**
  - **S/W Quality Risks**
  - **Metrics and Measures**
  - **Using Risk Assessments to Manage**
- 0 **Discussion**
  - **Components of a Risk Management Framework**
  - **Transferring to Industry and Academia**
  - **Including OO Specific Risks**
- 0 **Summary**

Direct and Indirect Technology Transfers of SQAE™ Methods and Tools

SQAE™ Licensees:

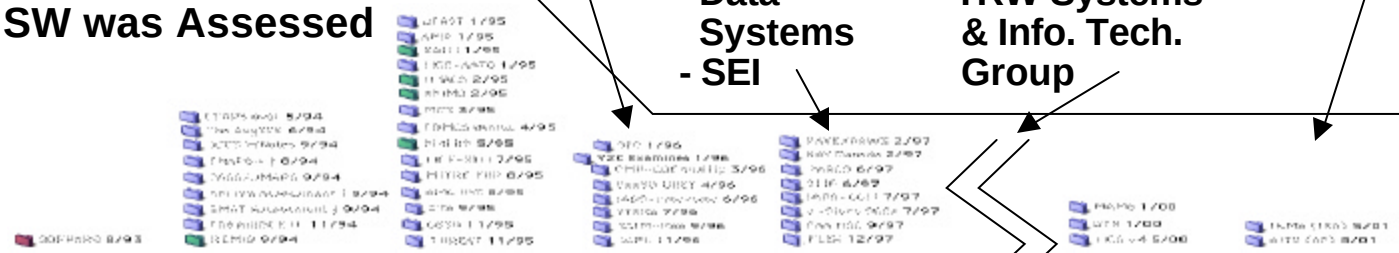
1996
- Mitretek

1997
- Lockheed Martin
- Grumman Data Systems
- SEI

1998
- General Dynamics Info. Systems
- TRW Systems & Info. Tech. Group

2001
- Ecole de technologie superieure

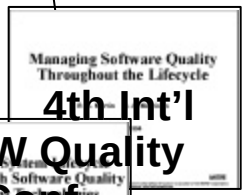
Organizations Who's SW was Assessed



1993 1994 1995 1996 1997 2000 2001 2002

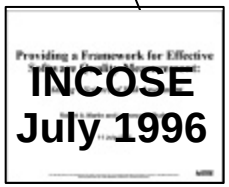
1998/1999

IEEE CS
May 1994



4th Int'l
SW Quality
Conf.
July 1994

STSC
April 1995



INCOSE
July 1996

SEI
SEPG
May 1996

STSC
April 1998

STSC
May 2002

SQAE ™
Presentations

Discussion Outline

- 0 **Introduction**
  - **Managing S/W Quality Issues by Measuring Risks**
- 0 **Background**
  - **S/W Quality Risks**
  - **Metrics and Measures**
  - **Using Risk Assessments to Manage**
- 0 **Discussion**
  - **Components of a Risk Management Framework**
  - **Transferring to Industry and Academia**
  - **Including OO Specific Risks**
- 0 **Summary**

Modifications to the SQAE™ to Deal with Object Oriented Design and Development *(1 of 2)*

Exercise A - The first exercise area concentrates on those activities that can be accomplished by examining two large functional areas of the code. The activities in this exercise are listed below.

	For non-OO design/implementations	For OO design/implementations
3.1 ~~Is the structure of the design hierarchical in a top down design within tasking threads?~~ Has the code been structured into cohesive classes? ***LOCM*** **Lack of Cohesion in Methods**	~~Using the two large functional areas of the code from 1.10, run McCabe BattleMap, Refine, or another tool to produce structure chart. Examine.~~ ~~Rating will be between 1 and 0, where 1 is the higher rating. 1 for crisp hierarchy, .6 for discernible hierarchy, 0 if neither.~~	Using the two large functional areas from 1.10, examine the classes and count the number of method pairs whose similarity is 0 and those that are not 0. Rating will be between 1 and 0, where 1 is the higher rating. 1 - 20 * the average ratio of for each class of the similar method pairs with 0 similarity minus those whose similarity is not 0.
3.2 ~~Do the functional groupings of units avoid calling units outside their functional area?~~ Has code been structured into classes to avoid excessive coupling between classes? ***CBO*** **Coupling Between Objects**	~~Ignoring utility and encapsulated service calls, examine the structure chart from 3.1.~~ ~~Rating will be between 1 and 0, where 1 is the higher rating. 1 for strong isolation, .6 for generally isolated, 0 if neither.~~	Examine the modules from 1.10, and count the number of classes to which the different classes are coupled to other classes through methods and/or instance variables. Rating will be between 1 and 0, where 1 is the higher rating. 1 - 20 * the average ratio of the classes in these modules coupled to other classes to total classes in the system.
3.10 ~~Has code been structured to minimize coupling to global variables?~~ Have classes been structured into methods to avoid having a large number of methods being invoked for each input message to a class object? ***RFC*** **Response For a Class**	~~Examine the modules from 1.10, using a variable utilization map (such as a Set-Use diagram) and calculate the ratio of global variables used versus the total number of variables used for each procedure.~~ ~~Rating will be between 1 and 0, where 1 is the higher rating. 1- 20 * the average ratio of global variables to total variables.~~	Examine the classes from the modules from 1.10, and count the number of methods that could be executed in response to a message received by an object of each class. Rating will be between 1 and 0, where 1 is the higher rating. 1 - 20 * the average ratio of the methods in a class invoked by a message to the class to the total number of methods in the class.

Modifications to the SQAE™ to Deal with Object Oriented Design and Development *(2 of 2)*

Exercise F - The sixth exercise area activities look over all of the code loaded for a variety of tasks. The activities in this exercise are listed below.

	For non-OO design/implementations	For OO design/implementations
7.2 ~~Is the source code of low complexity (e.g. McCabe Cyclomatic…)?~~ Are the classes of low complexity? ***WMC*** **Weighted Methods Per Class**	~~Using the available source code, calculate the Cyclomatic complexity.~~ ~~Rating will be between 1 and 0, where 1 is the higher rating. Calculate the average Cyclomatic complexities and standard deviation for all functions and procedures. Score will be 1 -[(average + s.d. - 15) x 0.02.]~~	Using the available code, examine the methods of each class and calculate the sum of the Cyclomatic complexities for each method in a class. Calculate the average and standard deviation of the class complexities. Rating will be between 1 and 0, where 1 is the higher rating. Score will be 1 -[(average + s.d. - 15) x 0.02.]
7.7 ~~Is the code segmented into procedure bodies that can be understood easily?~~ Has code been organized into classes and sub-classes that can be understood easily? ***NOC*** **Number of Children**	~~Using the available source code, examine the code.~~ ~~Rating will be between 1 and 0, where 1 is the higher rating. 1 for all procedure bodies are less than two pages @ 50 lines per page, .8 for 80% are less than two pages, .5 for 50% are less than two pages, and 0 otherwise.~~	Using the available code, count the number of direct sub-classes for each class. Rating will be between 1 and 0, where 1 is the higher rating. 1 - 20 * the average ratio of classes to the direct sub-classes.
7.9 ~~Have all procedures been structured to avoid *excessive* nesting?~~ Have classes been structured to avoid large levels of inheritance? ***DIT*** **Depth of Inheritance Tree of a Class**	~~Using the available source code, calculate the average and standard deviation of the nesting levels.~~ ~~Rating will be between 1 and 0, where 1 is the higher rating. Score will be the ratio of 13 / (average squared + one sigma squared) with a maximum score of 1~~	Using the available code, calculate the length of the longest path of inheritance to each module. Calculate the average and standard deviation of the inheritance lengths. Rating will be between 1 and 0, where 1 is the higher rating. Score will be 1 -[(average + s.d. - 15) x 0.02.]

Discussion Outline

- 0 **Introduction**
  - **Managing S/W Quality Issues by Measuring Risks**
- 0 **Background**
  - **S/W Quality Risks**
  - **Metrics and Measures**
  - **Using Risk Assessments to Manage**
- 0 **Discussion**
  - **Components of a Risk Management Framework**
  - **Transferring to Industry and Academia**
  - **Including OO Specific Risks**
- 0 **Summary**

Summary: The Value of the SQAE™

- Easy to learn and apply for experienced developers
- Can provide an independent, objective assessment with community norms of key metrics for comparison of a project with the practices of its peers.
- Follows a repeatable process
- Provides specific detail findings
- Minimal effort to accomplish (5 - 6 staff weeks per system)
  - How large the application is, the number of different languages used, and the type of assessment desired
  - A small level of additional effort is needed when we run into a language we have not previously encountered
- Framework for comparing and contrasting systems
- Provides mechanism for obtaining a “past performance” measure of contractors
- Brings out lifecycle concerns and issues
- Proven ability to adapt to technology changes and the changes in software development methodologies

Summary: The Value of a SQAE™ (Concluded)

- Can be used as a pro-active framework for stating quality reqt's
  - Many quality measures are easily restated as req'ts for coding & design stds
  - Can use as part of an award fee determination
  - SOW words to ensure Government has access to code and doc's
- A variety of Government customers have been interested in the continued application of our work
  - Augmentation of the SEI SCE to look at product as well as process
  - Supports Air Force "supportability assessment" task
  - Helping compare and contrast legacy systems
- Assessments have been consistent with "other" opinions including the developer's
- Can track assessed contractors through project milestones
  - Comparing risk drivers and risk mitigators experienced vs. Software Quality Assessment risk profile