

**IMPROVING
S/W QUALITY NORMS
WITHIN
MILITARY SYSTEMS**

*Some Ideas on Changes to the
DII COE S/W Quality Compliance Process*

Robert A. Martin and Audrey Taub

22 April 1998

DISCUSSION OUTLINE

- **Introduction**

- **What do we mean by quality?**

- **Background**

- **How can you measure S/W quality?**
- **What makes a usable quality assessment?**
- **What are the uses of S/W quality assessments?**

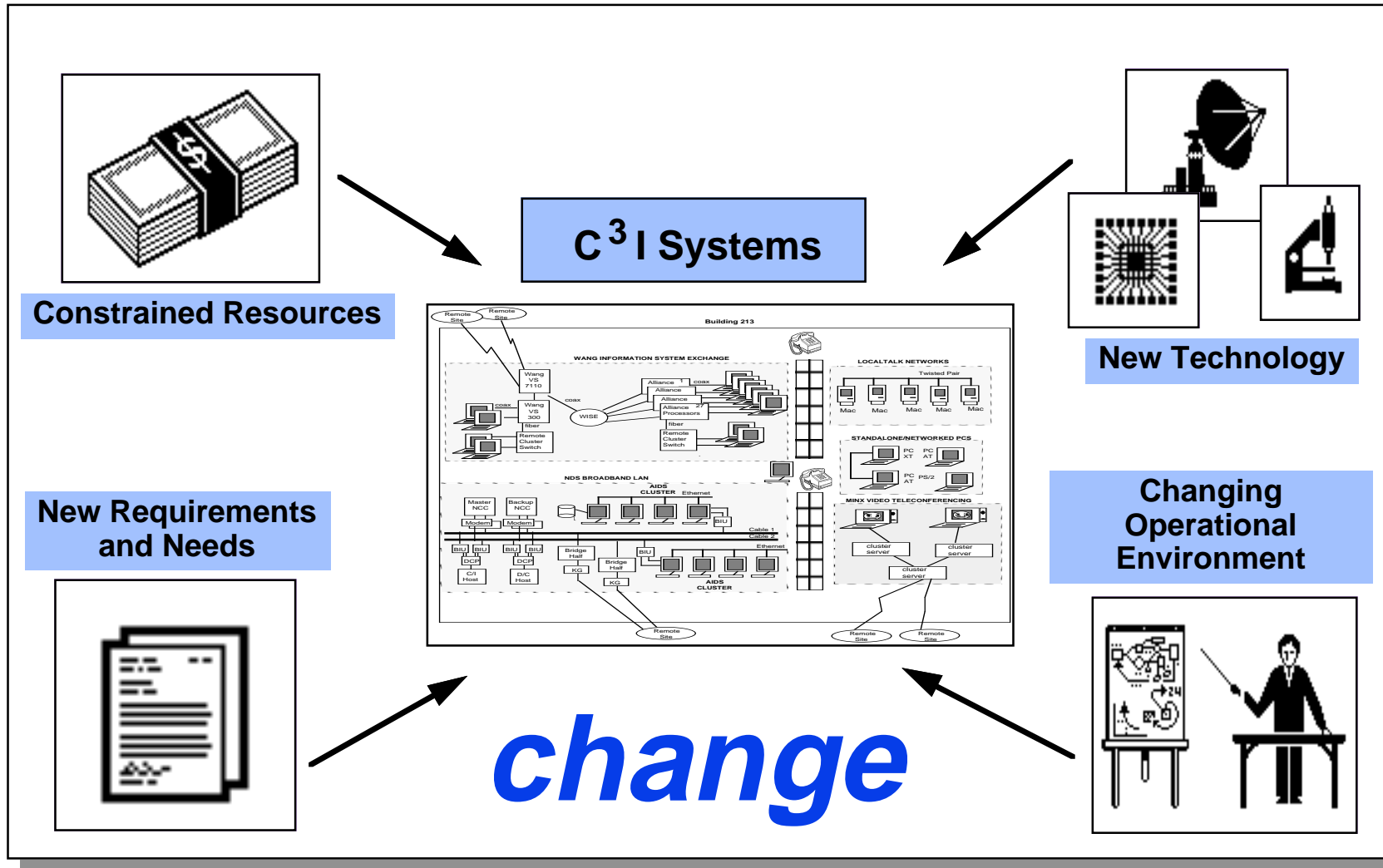
- **Discussion**

- **What does the DII COE S/W Quality Standard measure?**
- **What is missing or of questionable utility?**

- **Recommendations**

- **Constructing a more useful standard**
- **Impediments to implementation**

Why Are We Interested In Quality Anyway?



The Information Technology World Is Rapidly Changing

- IT organizations face many forces of change
 - Ever-changing user requirements
 - Hardware obsolescence rates that seem to accelerate
 - Commercial software is always being updated
 - Systems must be postured for growth and evolution
- Systems become “brittle” over time
 - Small changes can ripple through the system and incur unanticipated problems that increase cost and risk
 - Measurements for overall software “quality” are often neglected during development
 - Code structure, design, and the rationale for changes are lost

Software Quality Teams must do more than remove errors!

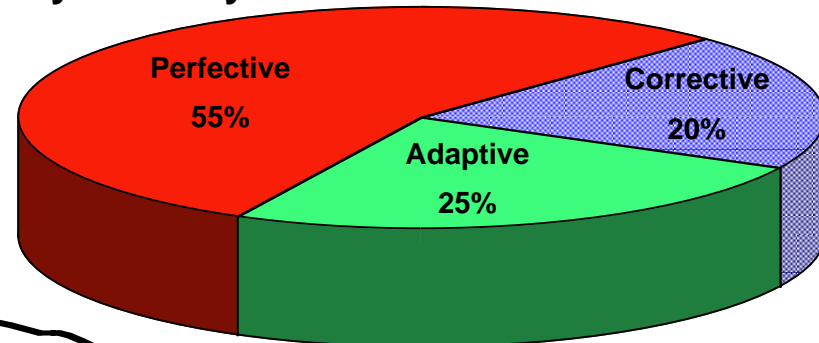
Looking Beyond Errors To Judge The Quality of Software

- Traditionally, most software organizations have focused on development
 - Manage schedule
 - Manage cost
 - Provide desired functionality to users
 - Maintainability issues are frequently deferred until the product is fielded
- Why focus on a lifecycle quality 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 the DOD spends on software goes toward maintenance

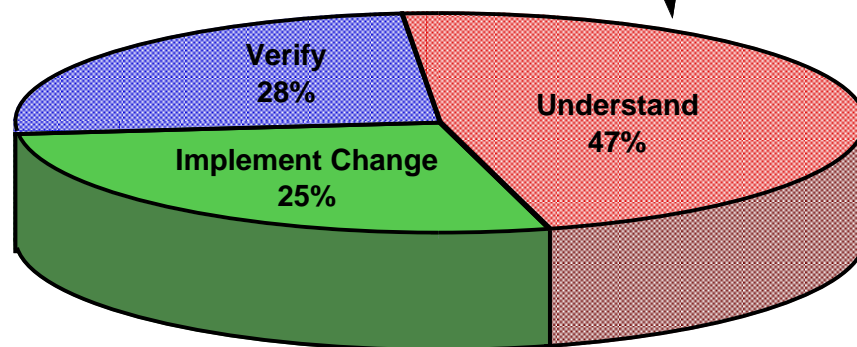
What is Software Maintenance?...and... What Do Software Maintainers Do?

A S/W maintainer's job activities over a 15 year lifecycle



(Lientz & Swanson Survey, 1980)

- Largest source of expenses in maintenance are driven by:
 - the time spent trying to understand the structure and behavior of the software



(McClure '90)

Why?

- Complexity coupled with lack of good supporting documentation

Defining Software Quality Issues

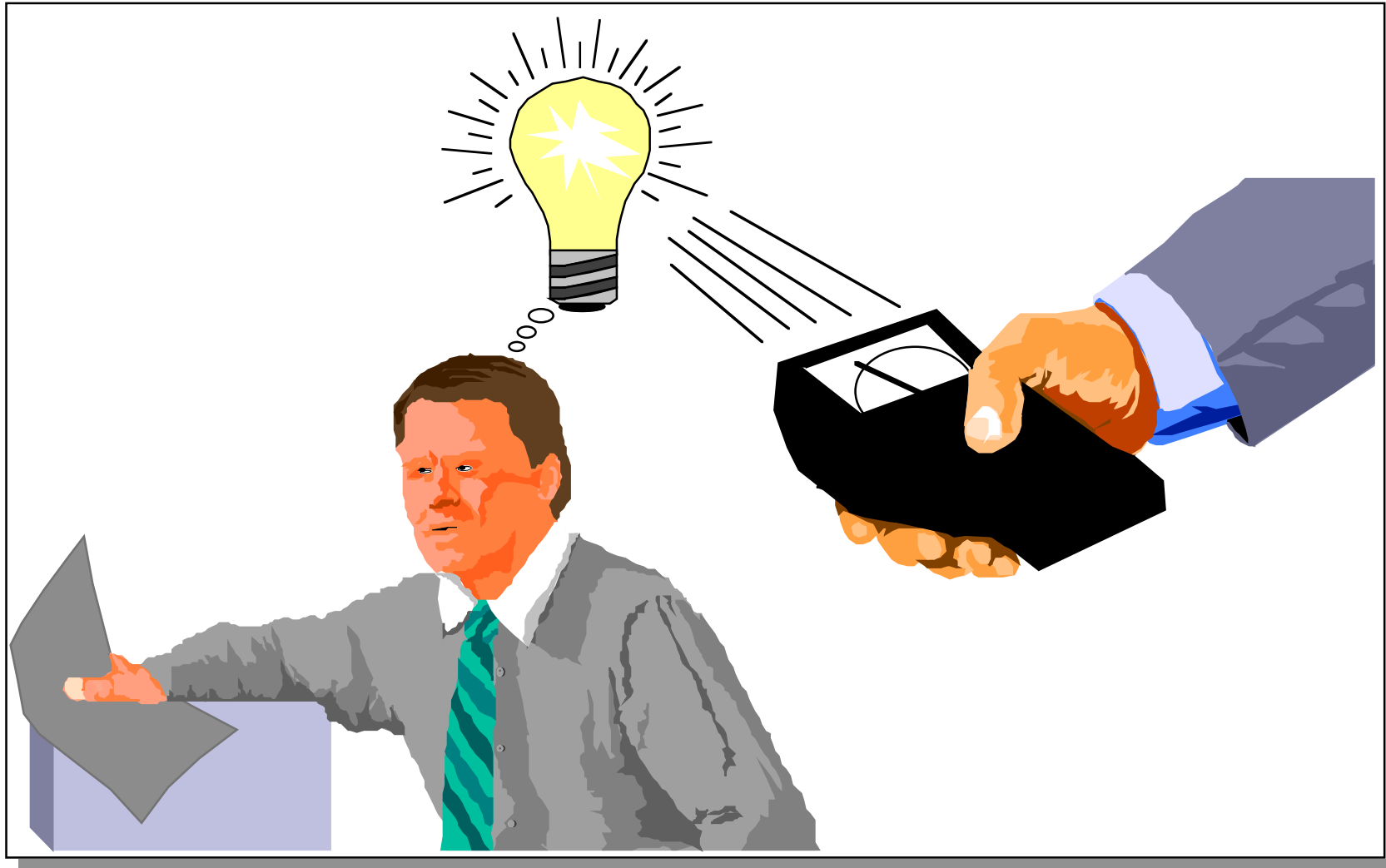
- What is the system I currently have really like?
 - Type & level of detail in the paper documentation
 - Understandability of the code and documentation
- What flexibility has been left to me by the original developers?
 - Hard coded assumptions or limitations
 - Overall capacity for functional growth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

In short, we need to know if the system will ever work again after we make a change!

DISCUSSION OUTLINE

- **Introduction**
 - What do we mean by S/W quality?
- **Background**
 - **How can you measure S/W quality?**
 - What makes a usable quality assessment?
 - What are the uses of S/W quality assessments?
- **Discussion**
 - What does the DII COE S/W Quality Standard measure?
 - What is missing or of questionable utility?
- **Recommendations**
 - Constructing a more useful standard
 - Impediments to implementation

How Do You Measure an Abstract Concept Like Quality

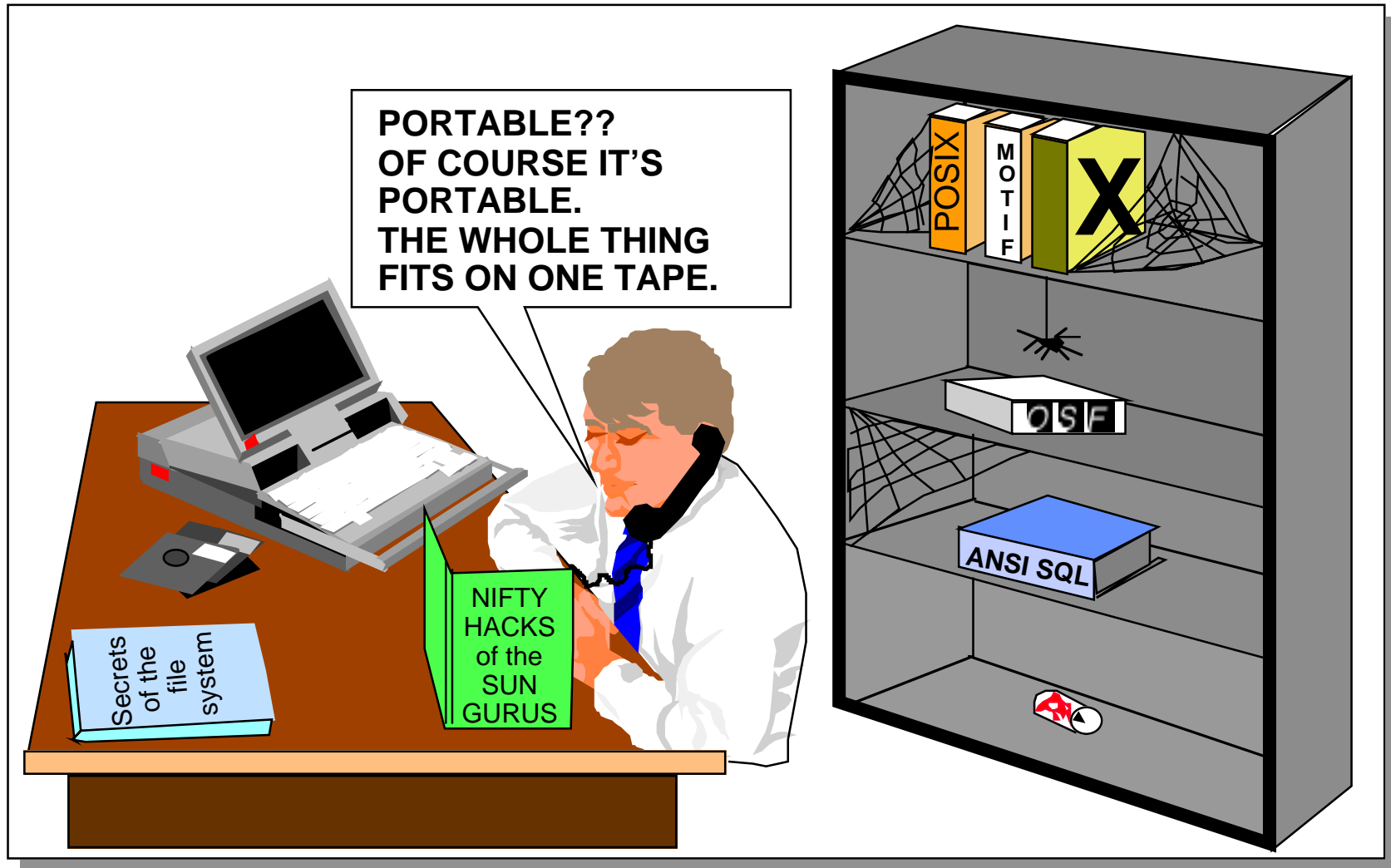


Software Lifecycle Quality Is Not Well Defined

- 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

Complications to a System's Software Quality

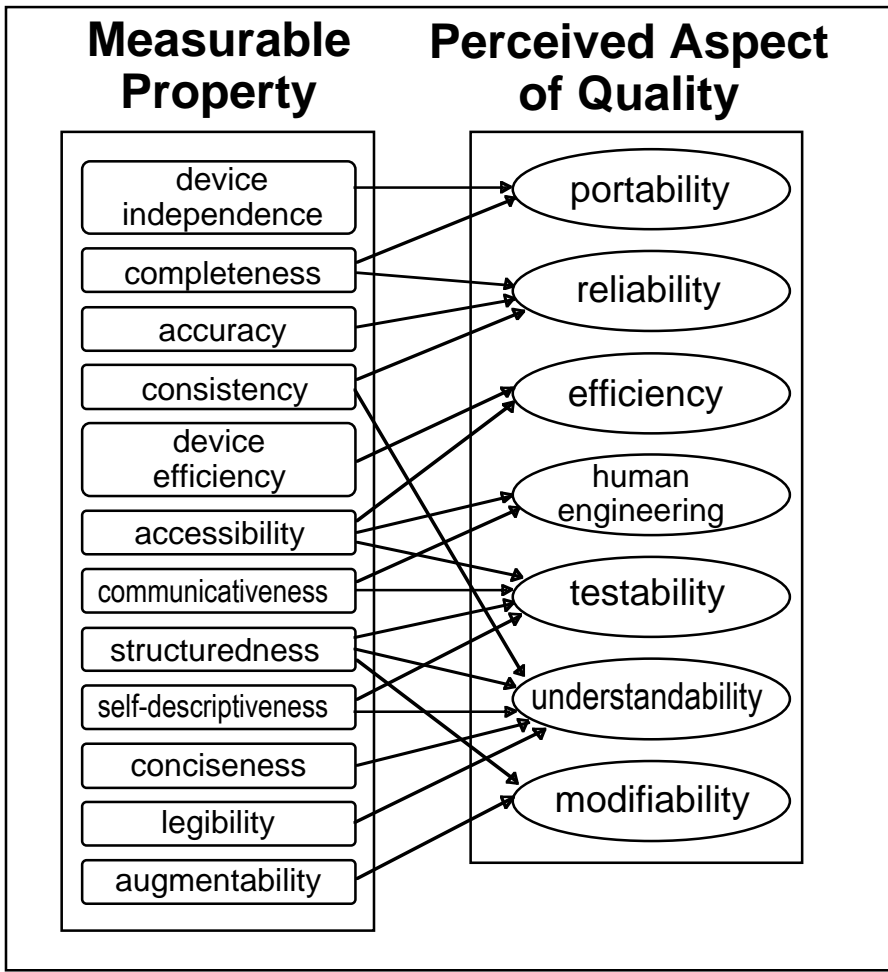


Establishing a Framework for Measuring Quality

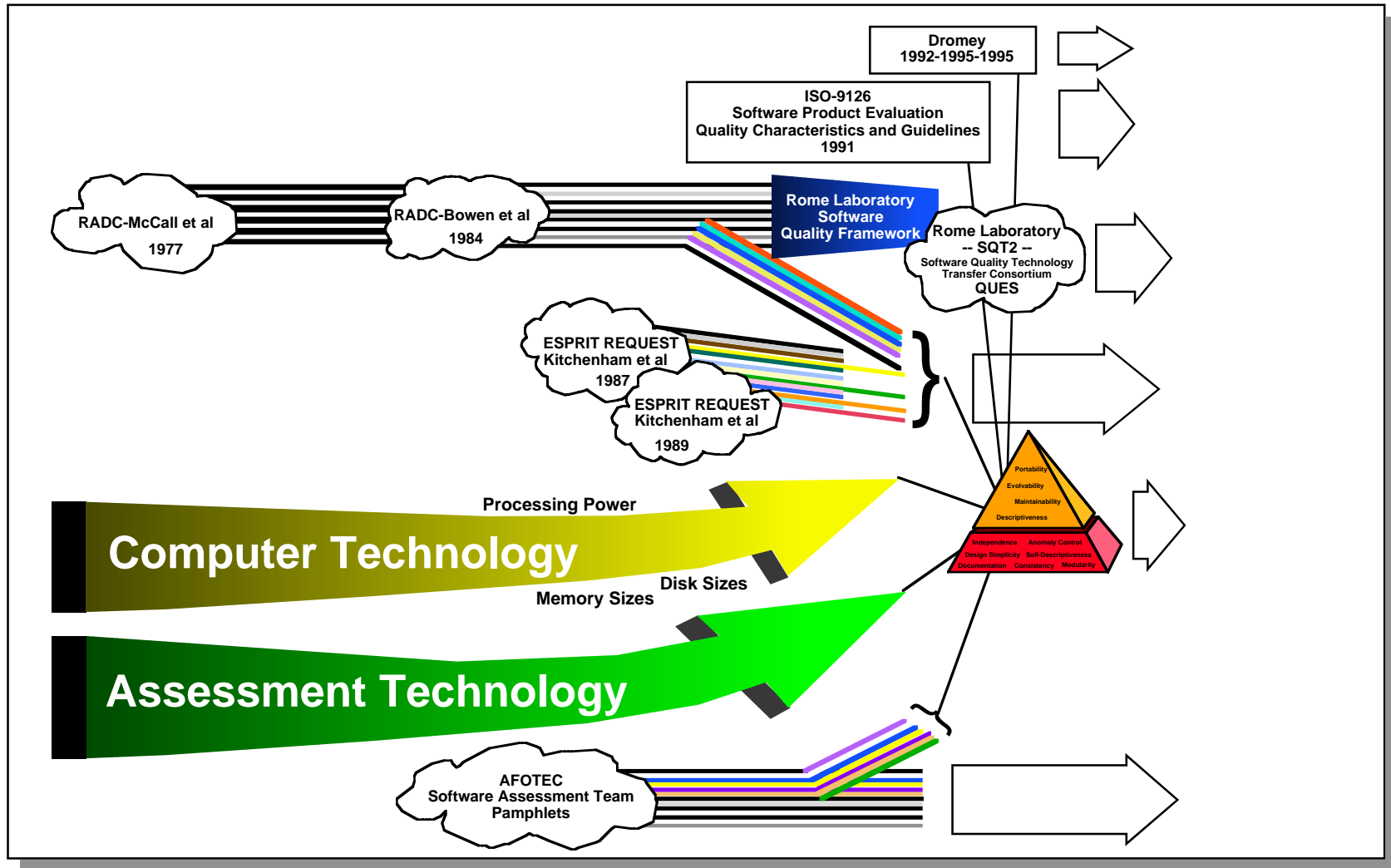
- **Many areas can help minimize a system's risks**
 - Some are well studied and have full fledged disciplines, technologies, and examination methodologies in place
 - Specifically: requirements traceability, functional completeness, and system testability are well established areas of study
- **The 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...

Basics of Quality Analysis Frameworks

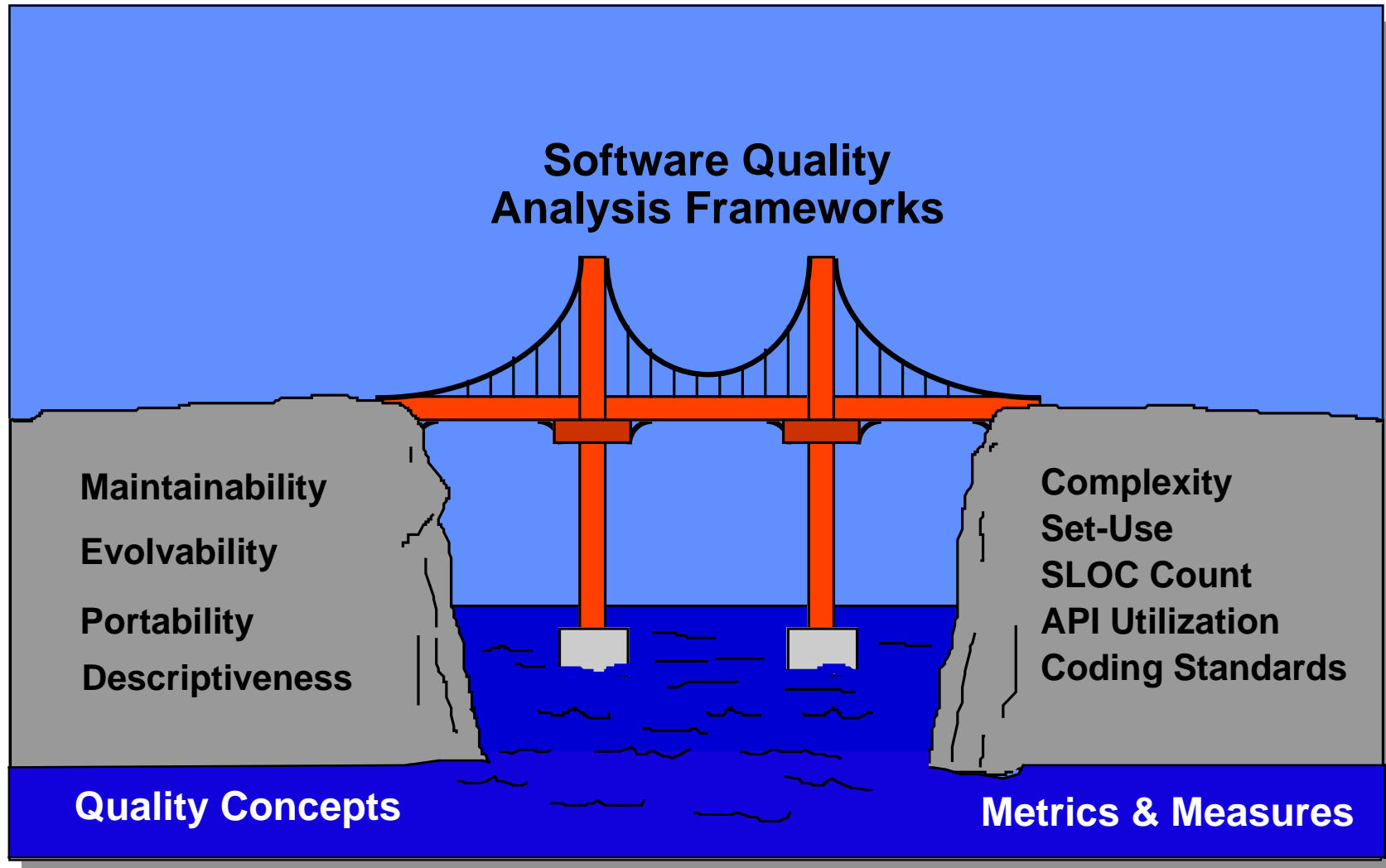
**Boehm et. al. RADC
Original Quality Factors
1977**



Relationships Among Software Quality Assessment Research Efforts



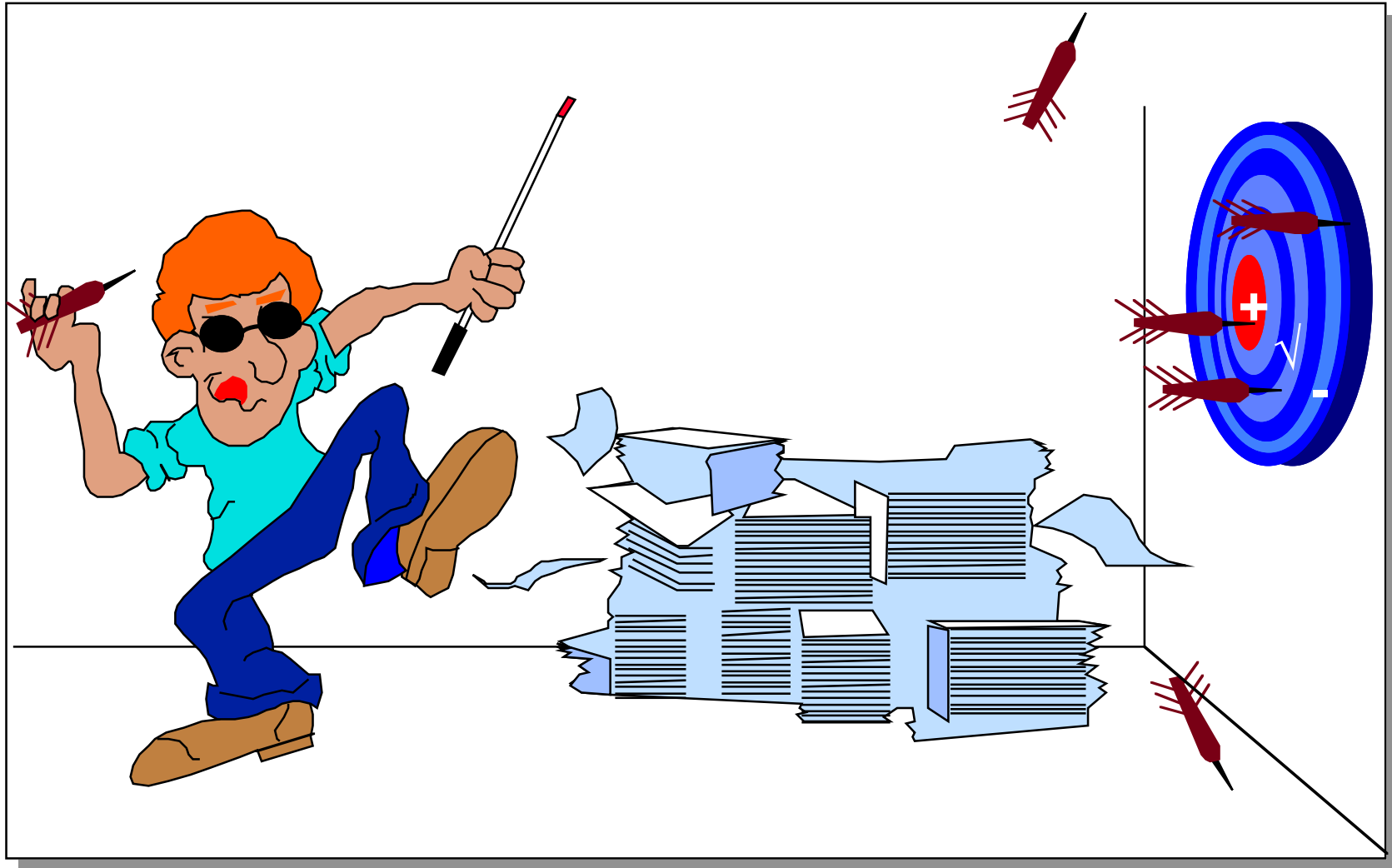
Bridging the Gap between The Measurable and Unmeasurable



DISCUSSION OUTLINE

- **Introduction**
 - What do we mean by S/W quality?
- **Background**
 - How can you measure S/W quality?
 - **What makes a usable quality assessment?**
 - What are the uses of S/W quality assessments?
- **Discussion**
 - What does the DII COE S/W Quality Standard measure?
 - What is missing or of questionable utility?
- **Recommendations**
 - Constructing a more useful standard
 - Impediments to implementation

One Method of Assessing Software Quality



Attributes Of A Useful S/W Quality 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**

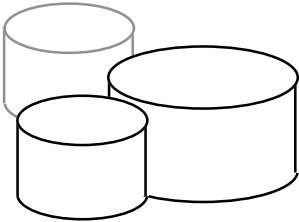
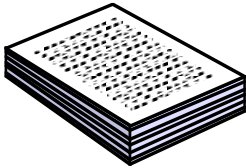
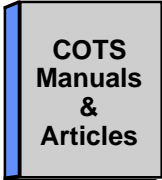
Quality assessment Finding Examples: Mitigators, Drivers, & Other Observations

<p>Risk Mitigators</p> <ul style="list-style-type: none"> ● 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. 	<p>Risk Drivers</p> <ul style="list-style-type: none"> ● 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
<p>Other Observations</p> <ul style="list-style-type: none"> ● 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 <ul style="list-style-type: none"> - The rest uses the proprietary DBMS calls ● Complete source code for gnu Perl was included as part of deliverable subsystem source code 	

This Chart Contains Representative Assessment Results

MITRE

Quality Assessment Foundation Examples

	Project A	Project B	●●● Project CZ	Total of Projects
Source Code 	112,000 LOC Ada, C, Shell, TAE+, SQL, X, MOTIF, Stored Procedures	558,000 LOC C, Shell, X, MOTIF	●●● 58,000 LOC ●●● Ada, C, ELF, ezX, SQL, X, MOTIF	51,173,315 LOC 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

This Chart Contains Representative Assessment Results



Examples of Tools Used in Assessing Software Quality

Code Quality Assessment Tools

The image displays several overlapping terminal windows from a Linux environment. The windows show the following content:

- Avatar 1 (Top Left):** Shows code quality metrics for 'avatar root 56' and 'avatar root 57'. It includes sections for 'STATS', 'Flow Complexity', 'Cyclomatic Complexity', and 'Flow Complexity Table'. The table shows a range of complexity values from 1 to 5, with a total of 3659.
- Avatar 1 (Middle):** Shows the 'mgrep' command being used to search for patterns. The output shows the usage and options for 'mgrep', including '-n', '-s', '-v', '-u', '-D', '-F', and '-L'.
- Avatar 1 (Bottom Left):** Shows the output of the 'mgrep' command, listing various file types and their counts.
- Avatar 1 (Bottom Right):** Shows the output of the 'mgrep' command, including the number of unique files affected (560).
- Avatar 1 (Far Right):** Shows a 'Select Report Dialog' box with a list of report types and a 'Please select tool:' field.
- Avatar 1 (Far Bottom Right):** Shows a woman sitting at a desk with a computer monitor, looking at the screen.

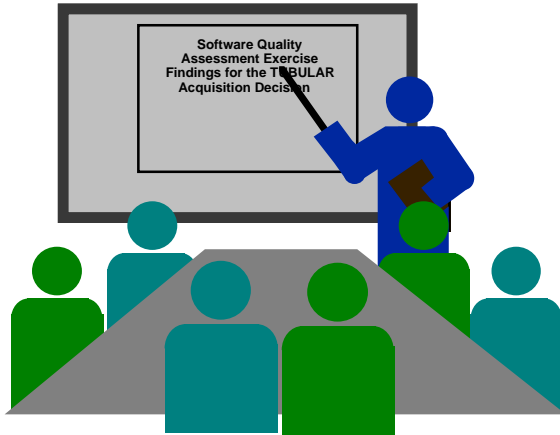
... many tools do not adequately address the use of commercial packages, or easily deal with multi-language applications, or help you correctly interpret their metrics.

DISCUSSION OUTLINE

- **Introduction**
 - What do we mean by S/W quality?
- **Background**
 - How can you measure S/W quality?
 - What makes a usable quality assessment?
 - **What are the uses of S/W quality assessments?**
- **Discussion**
 - What does the DII COE S/W Quality Standard measure?
 - What is missing or of questionable utility?
- **Recommendations**
 - Constructing a more useful standard
 - Impediments to implementation

Having An Understanding Software Quality Can Be Used In...

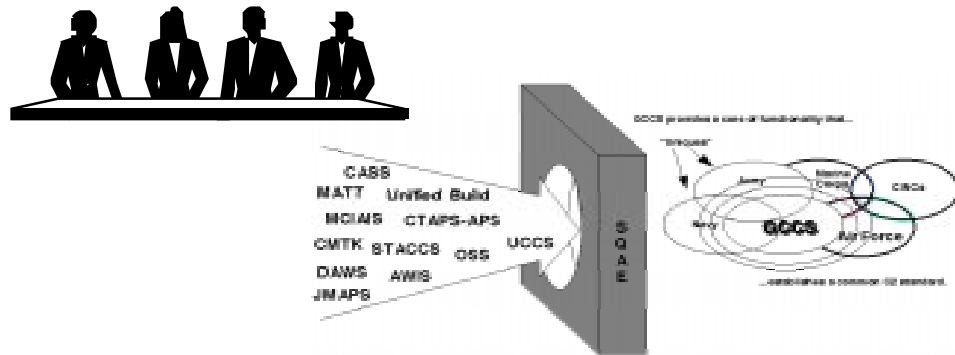
The Selection of Contractors



Reviews of S/W Releases for a Project Office



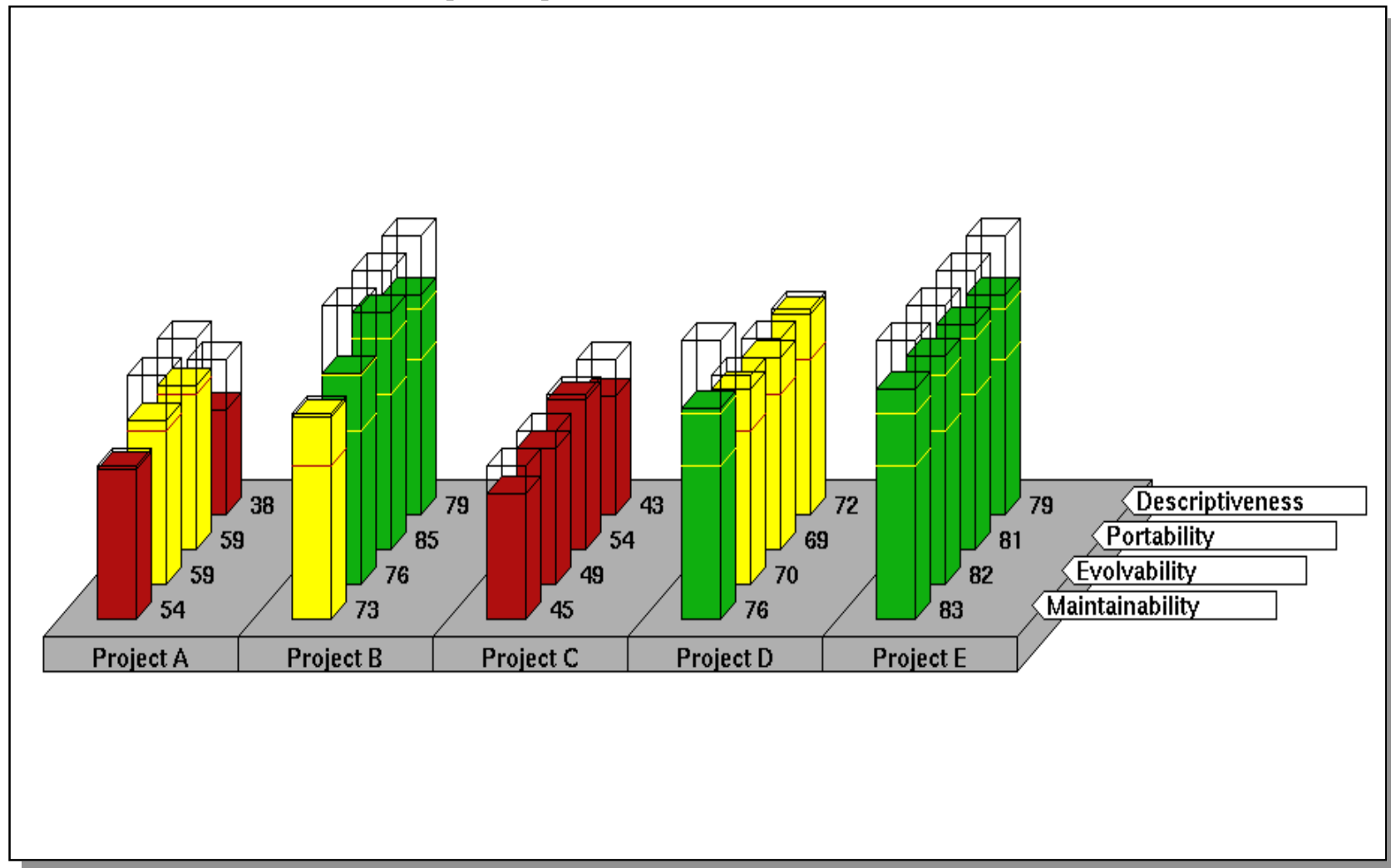
Selection of Migration Systems



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**

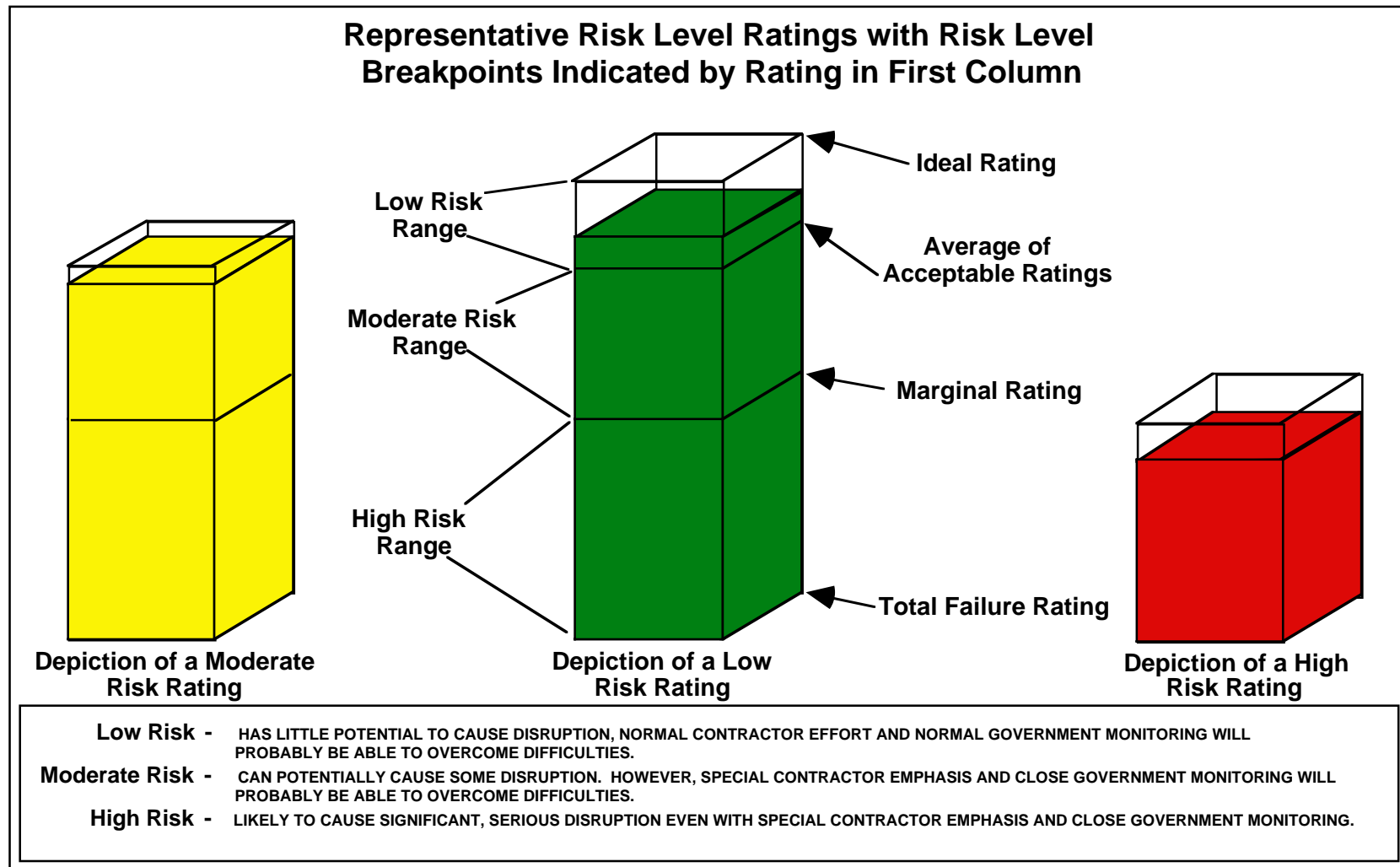
Examples of Software Quality Risk Profiles (3D)



This Chart Contains Representative Assessment Results

MITRE

One Way Of Indicating The Resulting Risk Profiles



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 (New Database, etc) code rewrites and unit tests are restricted to distinct areas rather than ripple through the 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

This Chart Contains Representative Assessment Results

MITRE

Assessment Reports And Systems Assessed To-Date

Program Risk Profile

Assessment Foundation

Examples of the Quality Component Assessment

The Augmented SCE Software Quality Assessment Exercise Methodology

Software Quality Assessment An Overview of the Evaluation Process and Methodology

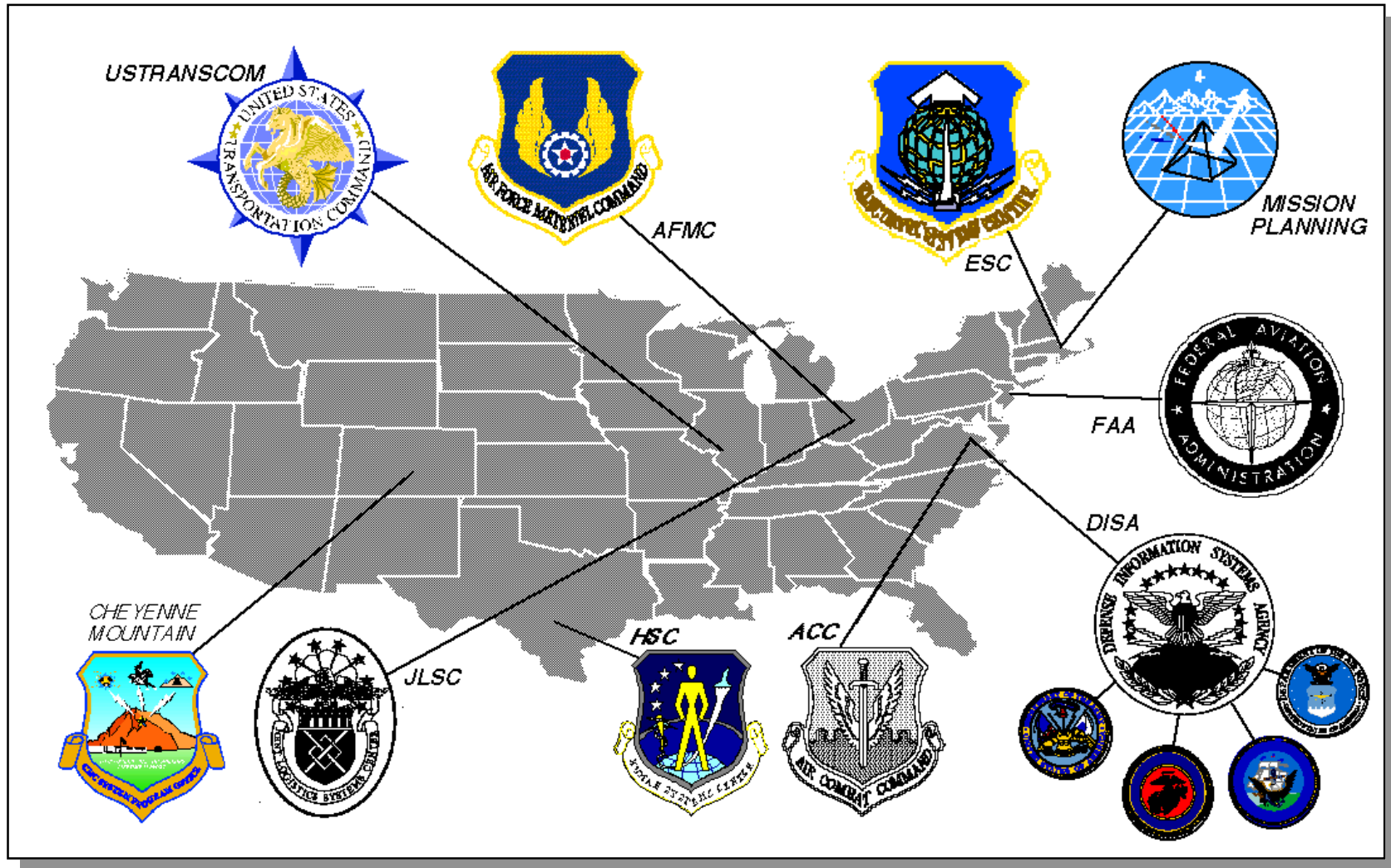
Project X
A Software Quality Assessment Exercise in Support of an Augmented SCE Debrief

Software Quality Assessment Exercise Report

Language	Percentage
C (ANSI and K&R)	52%
FORTRAN	6%
COBOL	7%
shell	2.5%
DB Information	4%
ORACLE	0.5%
INF-ORMIK	4.5%
GUI Languages	6%
SQL	3%
INGRESS	0.5%
others	3%
Ada	11%

Over 100 Systems and 51 Million Lines of Code Assessed

Software Quality Assessment Experience-Base



DISCUSSION OUTLINE

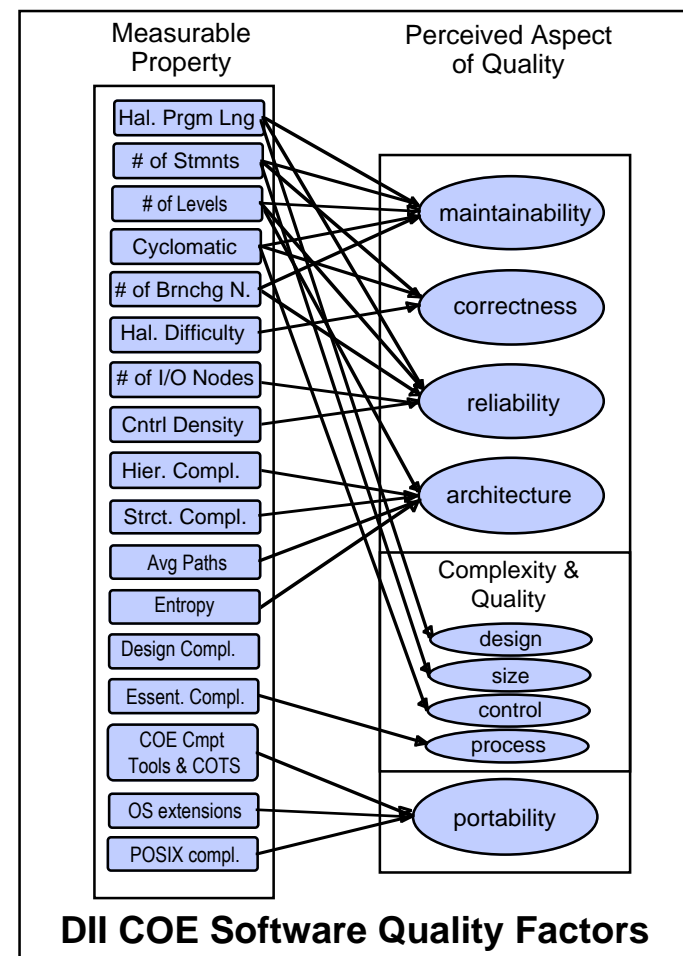
- **Introduction**
 - What do we mean by S/W quality?
- **Background**
 - How can you measure S/W quality?
 - What makes a usable quality assessment?
 - What are the uses of S/W quality assessments?
- **Discussion**
 - What does the DII COE S/W Quality Standard measure?
 - What is missing or of questionable utility?
- **Recommendations**
 - Constructing a more useful standard
 - Impediments to implementation

Stated Goals Of The DII COE Software Quality Compliance Process

- Directed at COE “common function” components
- Purpose is to:
 - Identify components that present significant risk factors in:
 - Integration
 - Maintainability
 - Correctness
 - Reliability
 - Identify cost effective candidates for renovation
 - Institutionalize software quality compliance assessment techniques within DII to manage costs and integration risks
 - Identify usage of non-public APIs
 - Increase testing effectiveness
 - Identify portability risks

DII COE S/W Quality Measurement Foundation And Focus

- Calculate risk rankings using equations and thresholds associated with fairly standard software metrics



COE Software Quality Metrics Definitions (1 of 2)

- **Halstead's**
 - **Length:** Measure of modularity of design
 - **Difficulty:** Measure of difficulty of developing the component
- **Cyclomatic Complexity:** Measures # of testable paths/component
- **Essential Complexity:** Measure of the structure of the testable paths in a component
- **Design Complexity:** Measures the complexity of the control flow implemented by the design
- **Source Lines of Code:** Physical length of a component
- **Control Density:** Measures percentages of control structures in a component
- **Max. # of Levels:** Measures depth of IF..THEN..ELSE Nests in components
- **Number of Branching Nodes:** Measures the # of “GO TOs” or number of abnormal exits from control structures and loops
- **Number of Input/Output Nodes:** Measures the number of ways in and out of a component

Note: These are ALL code-based metrics

MITRE

COE Software Quality Metrics Definitions (2 of 2)

- **Hierarchical Complexity:** A measure of the average number of components on a level
- **Structural Complexity:** Average number of calls per component in the call graph
- **Average Paths:** Average number of paths per node in the call tree
- **Number of Levels:** Number of levels in a class tree
- **Entropy:** Measure of orderliness in execution of the components in a call graph

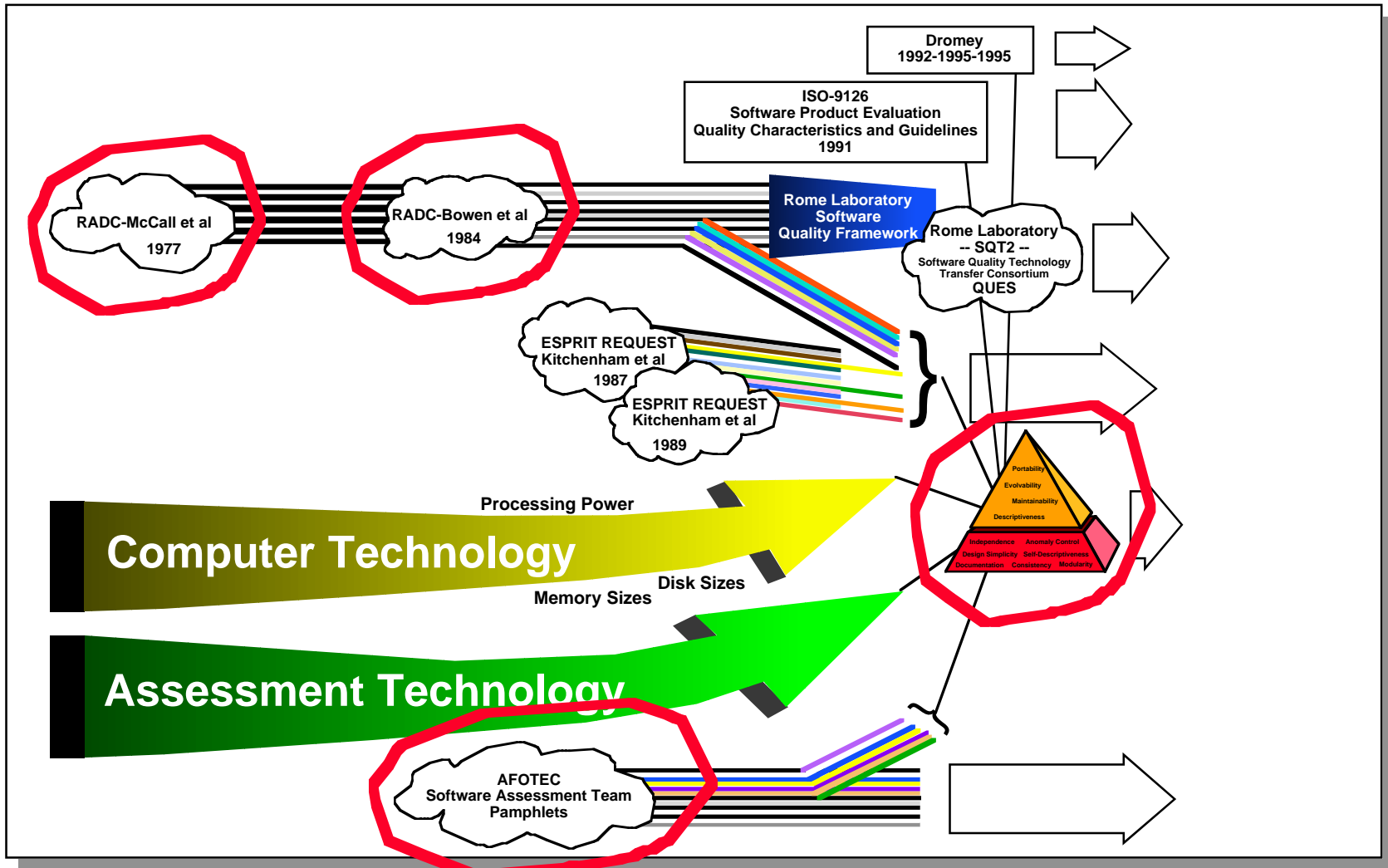
Note: These are also ALL code-based metrics

MITRE

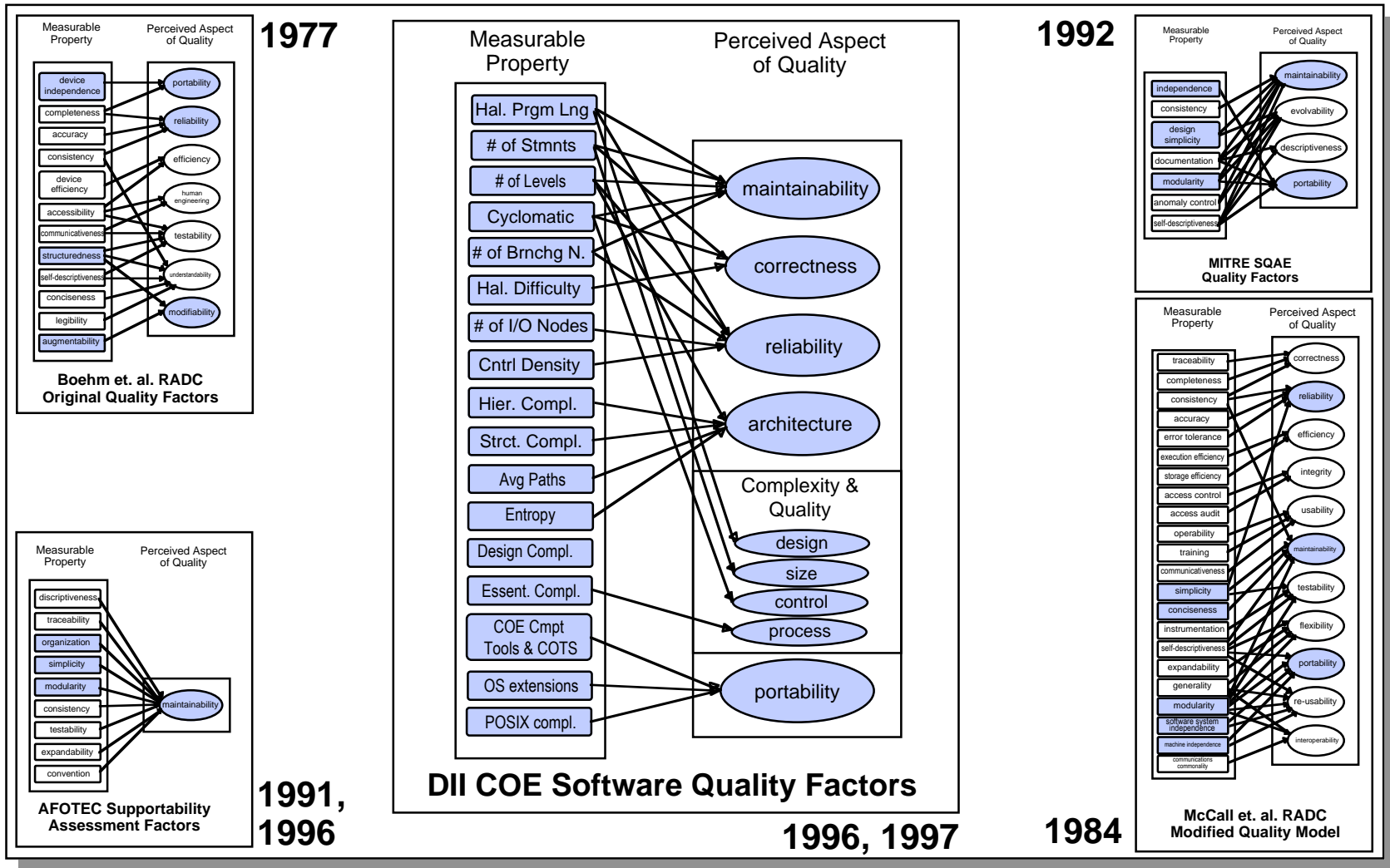
DISCUSSION OUTLINE

- **Introduction**
 - What do we mean by S/W quality?
- **Background**
 - How can you measure S/W quality?
 - What makes a usable quality assessment?
 - What are the uses of S/W quality assessments?
- **Discussion**
 - What does the DII COE S/W Quality Standard measure?
 - What is missing or of questionable utility?
- **Recommendations**
 - Constructing a more useful standard
 - Impediments to implementation

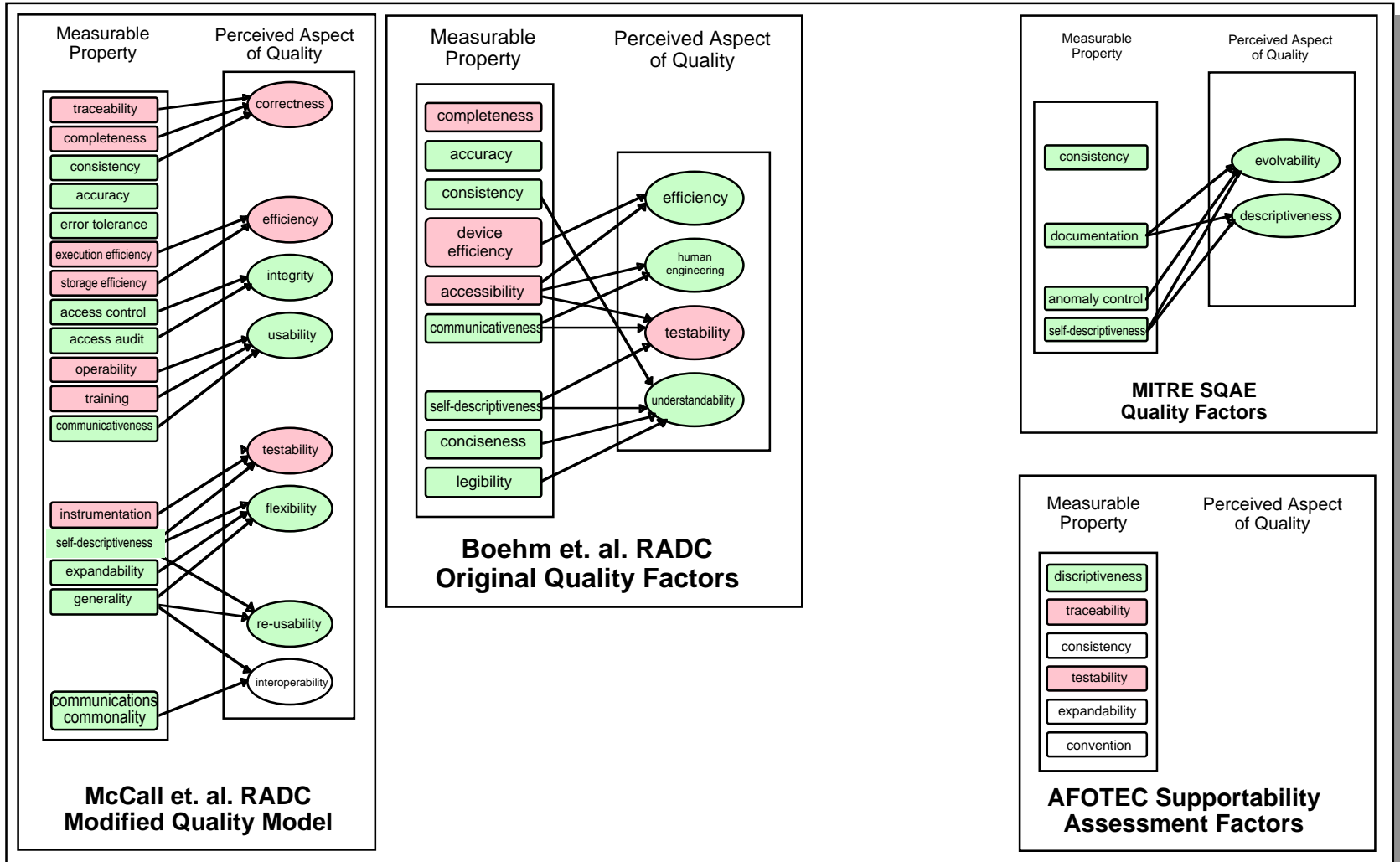
Defining An “Industry” For Comparing And Contrasting Purposes



Comparing Quality Coverage: "Industry" vs. DII COE S/W Quality



“Industry” Quality Concern Areas Not Covered By DII COE S/W Quality



Summary Of Significant Areas Missing From DII COE S/W Quality Assessment

- Risk areas assessments missing:
 - The availability and adequacy of design and coding standards and the software's adherence to these standards
 - The availability and adequacy of design documentation in both the words and the diagrams, as well as the programmer and user manuals
 - The adequacy of thorough prologs and comments
 - The understandability and intuitiveness of naming conventions and the adherence to them by the software
 - Adequate characterization and rationalization of dependence on COTS

Observations About The DII COE S/W Quality Assessment Process (1 of 2)

- It is systematic
- Calls for a large variety of analyses
- Produces pointed and direct conclusions
- Focuses on risks and addressing risks
- However, Assessment Reports are long and somewhat technical, requiring considerable understanding of computer software and the mechanisms and methods behind its development, as well as the application domain of the system (i.e., it is very hard to come away with an understanding of what needs to change and why)

Observations About The DII COE S/W Quality Assessment Process (2 of 2)

- Specifically, the items of concern here include:
 - Use of fixed thresholds independent of the application domain or the languages(s)
 - Use of too low (and too high) thresholds for metrics
- The values of the thresholds are not discussed or presented in any consistent fashion, nor are they defended. Essentially, there appears to be no justification for the values employed.
- There are no clear definitions to help distinguish between the implications of “reimplement” and “redesign” when applied to the problem code
- Some risks are not appropriately addressed. Specifically:
 - the use of COTS code generators and of COTS itself,
 - the adequacy of the documentation, comments, readability, and naming conventions used for the systems under assessment

DISCUSSION OUTLINE

- **Introduction**
 - What do we mean by S/W quality?
- **Background**
 - How can you measure S/W quality?
 - What makes a usable quality assessment?
 - What are the uses of S/W quality assessments?
- **Discussion**
 - What does the DII COE S/W Quality Standard measure?
 - What is missing or of questionable utility?
- **Recommendations**
 - **Constructing a more useful standard**
 - Impediments to implementation

Summary Recommendations (1 of 2)

- **Expand scope of assessment to address ALL COE-targetted applications, not just common ones**
- **Eliminate the dependence on fixed thresholds for all kinds of application domains and in all types of languages;**
 - **Take into account legitimate variations, such as appropriate use of case constructs in message processing code, which might need to be excluded from design complexity and cyclomatic ratings considerations;**
 - **Inspect listings and take into account the type of language before applying any SLOC judgments;**
- **Include the assessment of the source code that is fed into the automatic code generator since this input code is the material that will be maintained**
- **Clearly present and discuss the values of the thresholds**
- **Provide complete definitions as basis of terms used**
- **Consider the issues surrounding the use of transient COTS tools and applications over the lifecycle of the system**

Summary Recommendations (2 of 2)

- **Assess the availability and adequacy of design and coding standards and the software's adherence to these standards**
- **Assess the availability and adequacy of design documentation in both the words and the diagrams, as well as the programmer and user manuals**
- **Assess the adequacy of thorough prologs and comments**
- **Assess the understandability and intuitiveness of naming conventions and the adherence to them by the software**
- **Adequate characterization and rationalization of dependence on COTS**

Summary statement: Human judgment is still the last best recourse in understanding what we do with complicated software written for specific types of domains

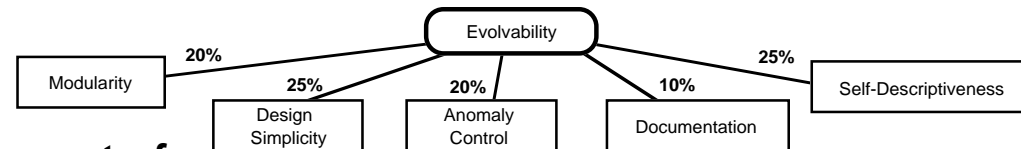
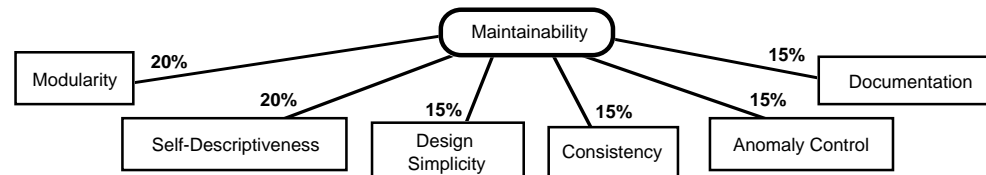
Guiding Principles: Breadth, Depth, and Repeatability

- 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 of development standards, ...)
 - Subjective measures (use of white space, usefulness of comments, level of design detail, ...)
- 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

Example: Software Quality Assessment Areas and Factors

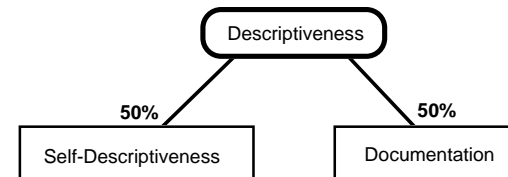
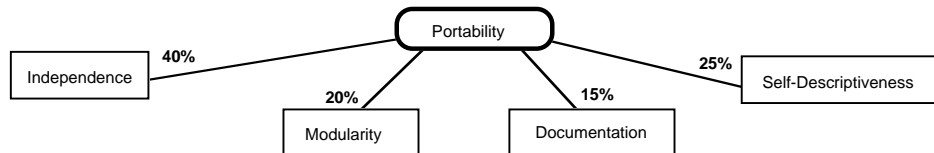
- Assess software against a defined set of quality areas:

- 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)



Examples of the Exercise Evaluation 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.

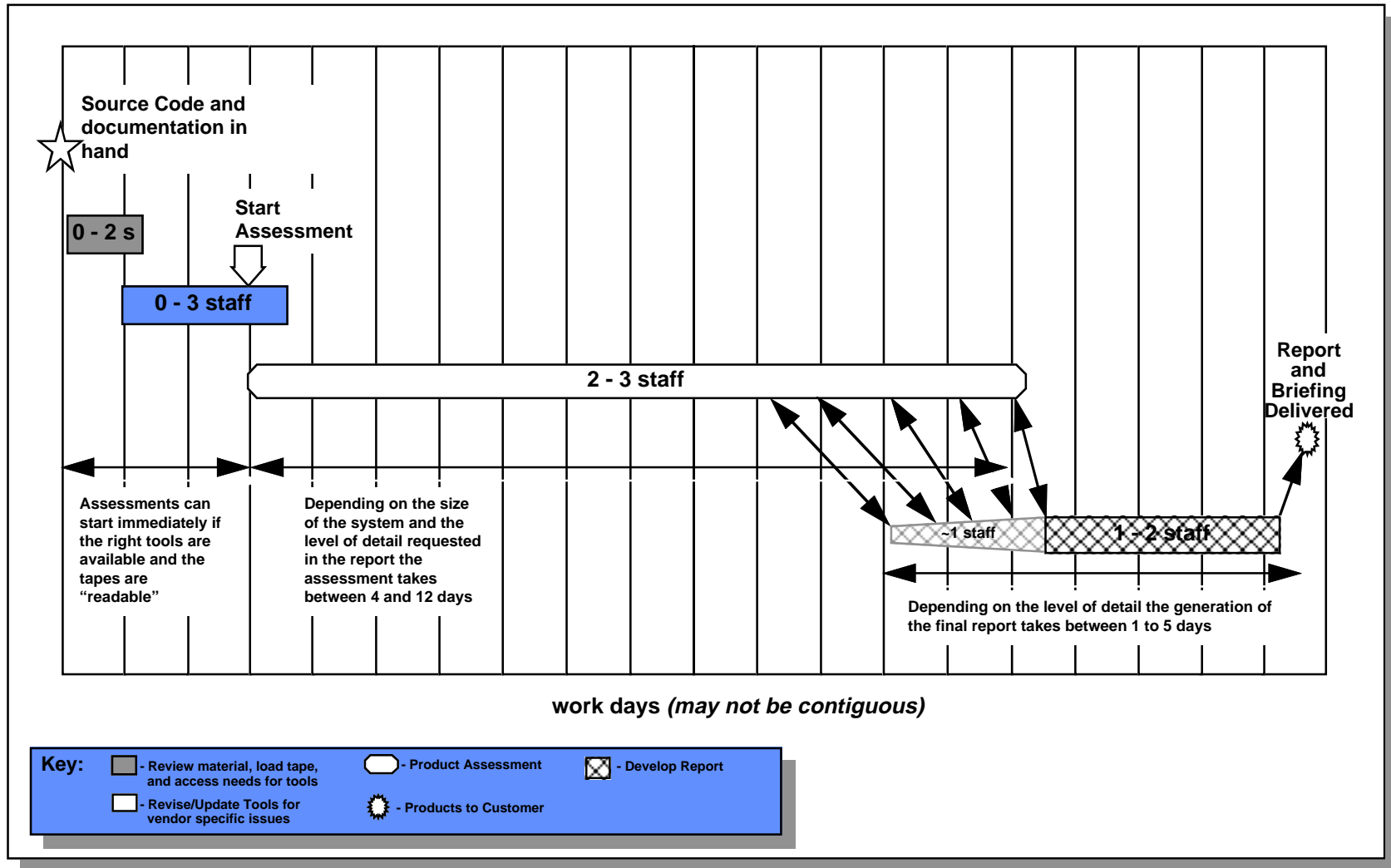
Rating will be between 1 and 0, where 1 is the higher rating. $1 - (\text{number of embedded dependencies} / \text{total number of dependencies})$



DISCUSSION OUTLINE

- **Introduction**
 - What do we mean by S/W quality?
- **Background**
 - How can you measure S/W quality?
 - What makes a usable quality assessment?
 - What are the uses of S/W quality assessments?
- **Discussion**
 - What does the DII COE S/W Quality Standard measure?
 - What is missing or of questionable utility?
- **Recommendations**
 - Constructing a more useful standard
 - Impediments to implementation

The Range of Software Quality Assessment Schedules

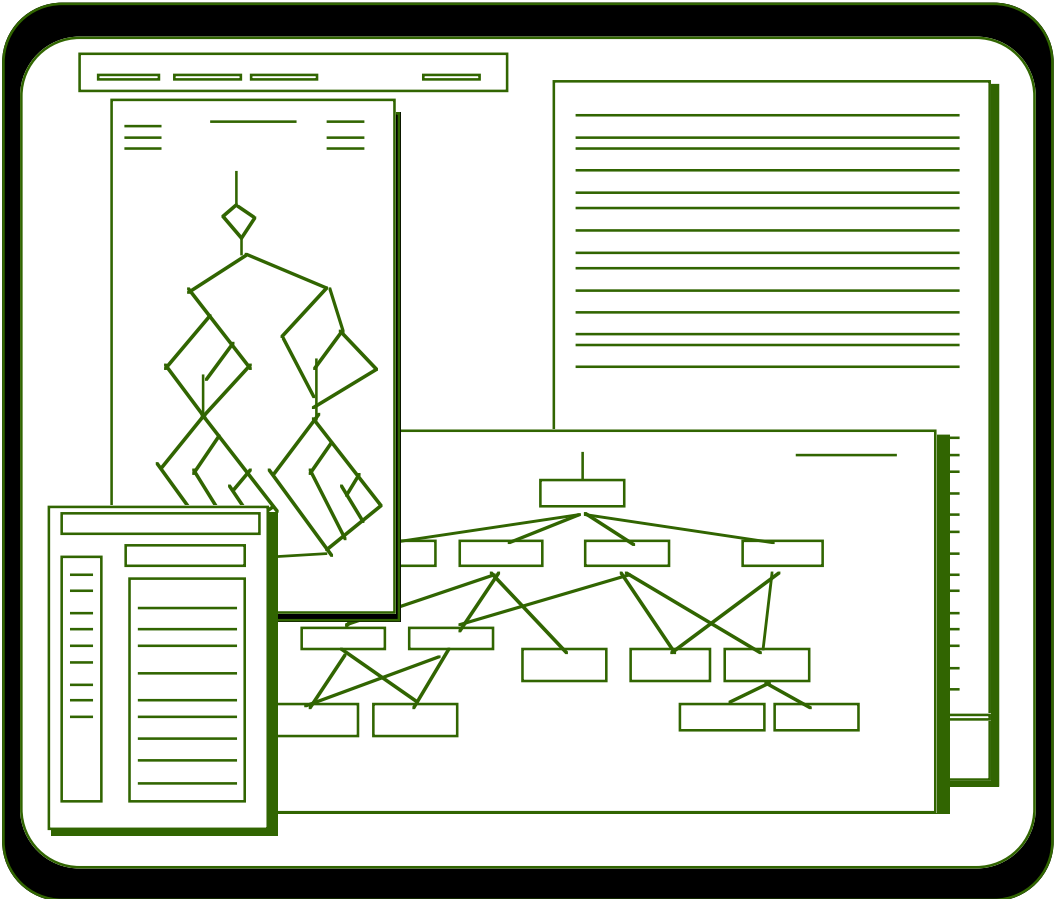


Need to Guide the Analyst with Reference Material


- To enhance consistency and repeatability we must provide adequate assistance and reference material



Tools That Can Handle Multiple Language Systems Are Needed



- Assess Code Structure
- Assess Code Complexity
- Find Patterns



Summary: The Value of a Software Quality Assessment

- **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**
- **Framework for comparing and contrasting systems**
- **Provides mechanism for obtaining a “past performance” measure of contractors**
- **Brings out lifecycle concerns and issues**