Managing Software Risks in Software Intensive Systems with Metrics and Measures

Robert A. Martin
13 April 2004
Discussion Outline

0 Introduction
- Managing S/W Quality Issues by Measuring Risks

0 Background
- S/W Quality Risks
- Metrics and Measures

0 Discussion
- Components of a SW Risk Management Framework
- Core SW Risks vs. Technology and Project Choices

0 Summary
- Using SW Risk Assessments to Manage
- Transferring to Industry and Academia
- Adapting SQAE to...
Introduction:
MITRE’s Software Risk Assessment Work

0 Wanted a framework for assessing lifecycle quality issues
0 Desired a flexible methodology for assessing risks in 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 risks
0 The resultant risk profiles have been used:
   - Selecting contractors based on quality of past efforts
   - Monitoring software quality during development
   - Identifying potential development process changes
   - Guiding future migration decisions
0 MITRE’s framework, methods, and tools have been proven
   - 110+ Systems, ~ 52 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
If, To Manage You Must Measure... How Do You Measure an Abstract Concept Like Software Quality?
One Method of Assessing Software Quality
Identifying Sources of Risks: Get Experts to Guide Through A Tool…

R. Williams & L. Bentrem (SEI)


http://www.sei.cmu.edu/products/events/acquisition/2004-presentations/
Discussion Outline

0 Introduction
   - Managing S/W Quality Issues by Measuring Risks
0 Background
   - S/W Quality Risks
   - Metrics and Measures
0 Discussion
   - Components of a SW Risk Management Framework
   - Core SW Risks vs. Technology and Project Choices
0 Summary
   - Using SW Risk Assessments to Manage
   - Transferring to Industry and Academia
   - Adapting SQAE to...
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

PORTABLE??
OF COURSE IT’S PORTABLE.
THE WHOLE THING FITS ON ONE TAPE.

MITRE
Quality Risks Related to Software Functionality & Technology Will Vary by Project - Others Don’t

0 System Functionality Dependent Risk Areas:
- Performance
- Accuracy
- Capacity
- Usability

0 System Technology Dependent Risk Areas:
- Architecture
- Design
- Secureness

0 Software Life-Cycle Risk Areas:
- Maintainability
- Portability
- Evolvability
- Documentation
# Targeted Attributes Of Our Software Quality Risk Assessment Methodology

- Repeatable (independent of the assessor(s))
- Independent of language, architecture, platform
- “Cheap” to perform
- Not dependent on presence of “all” code
- Provide detailed insight into the software risks
- Software centric
- Based on artifacts only
- 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
Establishing a Framework for Measuring Risks: Who Can We Borrow From?

- Many quality 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…
There Are Several Frameworks for Evaluating and Monitoring S/W Quality Risks

RADC-McCall et al 1977
RADC-Bowen et al 1984


Rome Laboratory Software Quality Framework

ESPRIT REQUEST Kitchenham et al 1987
ESPRIT REQUEST Kitchenham et al 1989

Drome

Computer Technology

Assessment Technology

Processing Power Memory Sizes Disk Sizes

USAF SSG SEP code review

AFOTEC Software Assessment Team Pamphlets

Rome Laboratory SQT2 Software Quality Technology Transfer Consortium QUES

MITRE
Discussion Outline

0 Introduction
   - Managing S/W Quality Issues by Measuring Risks

0 Background
   - S/W Quality Risks
     - Metrics and Measures

0 Discussion
   - Components of a SW Risk Management Framework
   - Core SW Risks vs. Technology and Project Choices

0 Summary
   - Using SW Risk Assessments to Manage
   - Transferring to Industry and Academia
   - Adapting SQAE to...
Metrics and Measures Guiding Principles: Breadth, Depth, and Repeatability

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

- **Step 1**: Identify risks to be evaluated.
- **Step 2**: Determine what is needed to evaluate the risk.
- **Step 3**: Determine whether the needed items are available; if items are not obtainable then discard that risk from the scope of the assessment.
- **Step 4**: Determine how the items will be examined to determine the risk information.
- **Step 5**: Determine how the risk information can be used to definitively identify risky versus mitigant items in the artifacts.
Discussion Outline

0 Introduction
   - Managing S/W Quality Issues by Measuring Risks
0 Background
   - S/W Quality Risks
   - Metrics and Measures
0 Discussion
   - Components of a SW Risk Management Framework
   - Core SW Risks vs. Technology and Project Choices
0 Summary
   - Using SW Risk Assessments to Manage
   - Transferring to Industry and Academia
   - Adapting SQAE to…
Bridging the Gap between The Measurable and Unmeasurable

Software Quality Analysis Frameworks

- Maintainability
- Evolvability
- Portability
- Descriptiveness

Quality Concepts

- Complexity
- Set-Use
- SLOC Count
- API Utilization
- Coding Standards

Metrics & Measures

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

Measurable Property | Perceived Aspect of Quality
---|---

Boehm et. al. RADC
Original Quality Factors 1977

Measurable Property | Perceived Aspect of Quality
---|---

McCall et. al. RADC
Modified Quality Model 1984

Measurable Property | Perceived Aspect of Quality
---|---

MITRE SQAE™
Quality Factors 1992 2003

Measurable Property | Perceived Aspect of Quality
---|---

AFOTEC Supportability Assessment Factors 1991 1996

- requirements issues
- testing issues

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

Measurable Property

- independence
- consistency
- design simplicity
- documentation
- modularity
- anomaly control
- self-descriptiveness

Perceived Aspect of Quality

- maintainability
- evolvability
- descriptiveness
- portability

MITRE SQAE™ Quality Factors
1992 - 2003

- software attributes

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

Components of the Software Quality Areas

- Portability
- Evolvability
- Maintainability
- Descriptiveness
  - Independence
  - Anomaly Control
  - Design Simplicity
  - Self-Descriptiveness
  - Documentation
  - Consistency
  - Modularity

Tools

Coding Standards, Software Development Plan, Design Documents, etc.
- SDP
- Coding Standards
- Detailed Design

Source Code and Listings
- (64 languages)

Specific Questions about the Code and Documentation

- (>110 systems)
- (52 million LOC)

Quality Risk Profile: Risk Mitigators
- [List of risk mitigators]

Quality Risk Profile: Risk Drivers
- [List of risk drivers]

Software Quality Risk Profiles
- [Graph or chart]

MITRE
Details: SQAE Areas and Factors

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

0 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.

#### Software System Independence

<table>
<thead>
<tr>
<th>Factor Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Does the software avoid all usage of specific pathnames/filenames?</td>
</tr>
<tr>
<td>2.2</td>
<td>Is the software free of machine, OS and vendor specific extensions?</td>
</tr>
<tr>
<td>2.3</td>
<td>Are system dependent functions, etc., in stand-alone modules (not embedded in the code)?</td>
</tr>
<tr>
<td>2.4</td>
<td>Are the languages and interface libraries selected standardized and portable? (i.e., ANSI…)</td>
</tr>
<tr>
<td>2.5</td>
<td>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)?</td>
</tr>
<tr>
<td>2.6</td>
<td>Is the generated code (i.e., GUI Builders) able to run without a specific support runtime component?</td>
</tr>
</tbody>
</table>

#### Machine Independence

<table>
<thead>
<tr>
<th>Factor Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7</td>
<td>Is the data representation machine independent?</td>
</tr>
<tr>
<td>2.8</td>
<td>Are the commercial software components available on other platforms in the same level of functionality?</td>
</tr>
</tbody>
</table>

### 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.

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

Consistency
Consistency is a factor that impacts nearly all quality issues and is a direct reflection of the policies and procedures for all aspects of the development process. Consistent software is built when there is a standards document and development is carried out in conformance with the document. Any potential issues on documentation, V&V protocols, data definition and nomenclature, etc., are identified and resolved.

Exercise A
The first exercise area concentrates on those activities that can be accomplished by examining the mission scheduler portion of the code and one other functional area of the code. The activities in this exercise are listed below.

Exercise B
The second exercise area concentrates on those activities that can be accomplished by examining the documentation on the system. The activities in this exercise are as listed below.

Exercise C
The third exercise area concentrates on those activities that can be accomplished by examining the largest module in each main execution thread. The activities in this exercise are listed below.

Exercise D
The first exercise area concentrates on those activities that can be accomplished by examining the mission scheduler portion of the code and one other functional area of the code. The activities in this exercise are listed below.

Exercise E
The second exercise area concentrates on those activities that can be accomplished by examining the documentation on the system. The activities in this exercise are as listed below.

Exercise F
The third exercise area concentrates on those activities that can be accomplished by examining the largest module in each main execution thread. The activities in this exercise are listed below.

Independent Independence comprises two broad groups: software system independence and machine independence. Here the issue is not to be the system to any specific host environment, which would require it to be impossible to migrate, evolve, or enhance the system.

Software System Independence
2.1 - Does the software use standard OS services? Between them and cross reference them.
2.2 - Is the software free of machine, OS and vendor specific extensions? Between them and system libraries and known vendor extensions.
2.3 - Are system dependent functions, e.g., standard library routines standardized and portable? Application programs should not be dependent on any one library or toolkit. Does the software have a need for any unique compilation in order to run (e.g., a 32-bit processor to "tweak" the code)?
2.4 - Does the generated code (e.g., GUI Builder) stand alone without a specific runtime environment?
2.5 - Is the data representation machine independent?
2.6 - Are the commercial software components available or other platforms in the same level of functionality?
### 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.

<table>
<thead>
<tr>
<th>Exercise</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.10</td>
<td>Are the naming conventions consistent for functional groupings?  &lt;br&gt;Examine the scheduling modules and one other large functional grouping and cross reference between them.  &lt;br&gt;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).</td>
</tr>
<tr>
<td>2.2</td>
<td>Is the software free of machine, OS and vendor specific extensions?  &lt;br&gt;Examine two large functional groupings of code and cross reference between them and system libraries and known vendor extensions.  &lt;br&gt;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.</td>
</tr>
<tr>
<td>2.3</td>
<td>Are system dependent functions, etc., in stand-alone modules (not embedded in the code)?  &lt;br&gt;Examine all known instantiations OS and vendor specific dependencies for encapsulation/isolation.  &lt;br&gt;Rating will be between 1 and 0, where 1 is the higher rating. 1 - (number of embedded dependencies/total number of dependencies)</td>
</tr>
</tbody>
</table>
Example of Risk Question Scoping and Guidance Provided to the Evaluator

Question: 6.2

Attribute: Is the vendor’s standard implementation of error handling consistently applied?

Factor: Anomaly Control

Contribution Level: 150

Rationale: Error handling should clearly indicate where and to whom the responsibility for a detected error is passed (to the user, to another process, etc.). This question does not penalize the vendor if not-published error handling standard is found. It is intended to establish the presence of a consistent application of error handling.

Precedence

Evaluation Scope: Search for error handling should be made on all code.

Evaluation Method:

This is a language specific question and may require reference to manuals. Ada satisfies this criterion by default by making handling explicit, except that scoring allows the same control issues as global variables to be used in doing any evaluation. For “C”, run grep with ‘err’ as the first argument and the grouping of code as the second. Try different cases. For Ada, search for “exception” and check that surrounding code clearly distinguishes local and non-local handling. Also, locate any type of vendor error handling documentation and search accordingly for appropriate intent and rationale.

Rating to be between 1 and 0, Where 1 is the higher rating. 1 for a match, 0.6 if consistent but no standard defined, 0 if neither.
Discussion Outline

0 Introduction
  - Managing S/W Quality Issues by Measuring Risks
0 Background
  - S/W Quality Risks
  - Metrics and Measures
0 Discussion
  - Components of a SW Risk Management Framework
  - Core SW Risks vs. Technology and Project Choices
0 Summary
  - Using SW Risk Assessments to Manage
  - Transferring to Industry and Academia
  - Adapting SQAE to…
Risks Related to Software Functionality & Technology Will Vary - Others Don’t

<table>
<thead>
<tr>
<th>System Functionality Dependent Risk Areas:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Performance</td>
</tr>
<tr>
<td>- Accuracy</td>
</tr>
<tr>
<td>- Capacity</td>
</tr>
<tr>
<td>- Usability</td>
</tr>
</tbody>
</table>

Assess “functional families of systems” for a range of “risk” appropriate to the family.

<table>
<thead>
<tr>
<th>System Technology Dependent Risk Areas:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Architecture</td>
</tr>
<tr>
<td>- Design</td>
</tr>
<tr>
<td>- Secureness</td>
</tr>
</tbody>
</table>

Assess design & architecture implementation risks - support issues...

Assess secureness of S/W (vuln mngt...)  

<table>
<thead>
<tr>
<th>Software Life-Cycle Risk Areas:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Maintainability</td>
</tr>
<tr>
<td>- Portability</td>
</tr>
<tr>
<td>- Evolvability</td>
</tr>
<tr>
<td>- Documentation</td>
</tr>
</tbody>
</table>
Bridging the Gap between The Measurable and Unmeasurable: Adding Secureness…

Software Quality Analysis Frameworks

Maintainability
Evolvability
Portability
Descriptiveness
Secureness

Quality Concepts

I/O handling
Complexity
Set-Use
SLOC Count
API Utilization
Coding Standards
Vuln Mngt

Metrics & Measures

MITRE
Discussion Outline

0 Introduction
   - Managing S/W Quality Issues by Measuring Risks

0 Background
   - S/W Quality Risks
   - Metrics and Measures

0 Discussion
   - Components of a SW Risk Management Framework
   - Core SW Risks vs. Technology and Project Choices

0 Summary
   - Using SW Risk Assessments to Manage
   - Transferring to Industry and Academia
   - Adapting SQAE to…
Having An Understanding Software Quality Risks Can Be Used In...

**The Selection of Contractors**

**Reviews of S/W Releases for a Project Office**

OUR ASSESSMENT OF YOUR SOFTWARE CAME UP WITH SEVERAL SUGGESTIONS ON RISK AREAS YOU CAN ADDRESS. THESE SHOULD MAKE THE SYSTEM CHEAPER TO MAINTAIN.

**Selection of Migration Systems**

MITRE
Software Quality Assessment Uses

Understanding the Software’s quality risks 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
## SQAE Informational Foundation

<table>
<thead>
<tr>
<th>Source Code</th>
<th>Written Material</th>
<th>Reference Material</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ada, C, Shell, TAE+, SQL, X, MOTIF, Stored Procedures</td>
<td>C, Shell, X, MOTIF</td>
<td>Ada, C, ELF, ezX, SQL, X, MOTIF</td>
</tr>
<tr>
<td>COTS Manuals &amp; Articles</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Project A
- 112,000 LOC
- Ada, C, Shell, TAE+, SQL, X, MOTIF, Stored Procedures

### Project B
- 558,000 LOC
- C, Shell, X, MOTIF

### Project CZ
- 58,000 LOC
- Ada, C, ELF, ezX, SQL, X, MOTIF

### Total of Projects
- 51,173,315 LOC
- Ada, C, FORTRAN, COBOL, shell, TAE+, SQL, X, MOTIF, UIL, Stored Procedures, GEL, ELF, ezX, ...

### References
- Product Literature
- Design and Code Stnds
- Reference Manual
- Users Manual

---

This Chart Contains Representative Assessment Results
## SQAE Finding Examples:
### Mitigators, Drivers, & Other Observations

<table>
<thead>
<tr>
<th>Risk Mitigators</th>
<th>Risk Drivers</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 Naming conventions used for modules and variables helps understand the code’s functionality.</td>
<td>0 Level of isolation and encapsulation of dependencies on platform and COTS packages varies between programmers</td>
</tr>
<tr>
<td>0 Good use of white space and indentation.</td>
<td>0 Use of environmental variables is undocumented and inconsistently done</td>
</tr>
<tr>
<td>0 Modules are easily viewed at once (&lt; 100 LOC)</td>
<td>0 Lack of written standards for naming conventions, error handling, data definitions, etc</td>
</tr>
<tr>
<td>0 Good functional documentation with high-level design.</td>
<td>0 Lack of standards for naming conventions, error handling data definitions, I/O, etc</td>
</tr>
<tr>
<td>0 Good design documentation, showing data and control flows.</td>
<td>0 Design documentation is poorly organized, incomplete, and at a very high level</td>
</tr>
<tr>
<td>0 Good developer documentation for supported APIs.</td>
<td>0 No low-level design information or functional allocation of software in documentation</td>
</tr>
<tr>
<td>0 Good top-down hierarchical structure to code.</td>
<td>0 Machine generated code documentation is inconsistent with the developed code documentation</td>
</tr>
<tr>
<td>0 Modules use straightforward algorithms in a linear fashion.</td>
<td>0 Machine generated code is undocumented</td>
</tr>
<tr>
<td>0 System dependencies are to readily available COTS software.</td>
<td>0 Procedure and file names depend on path for uniqueness</td>
</tr>
<tr>
<td>0 Code is of low complexity.</td>
<td>0 Hard coded absolute filenames/paths used</td>
</tr>
<tr>
<td>0 Logic flow through individual procedures is easy to follow.</td>
<td>0 UNIX commands hardcoded in the code</td>
</tr>
<tr>
<td>0 Disciplined coding standards followed by the programmers.</td>
<td>0 Hard coded variables used when symbolic constants should have been used</td>
</tr>
<tr>
<td>0 Considerable effort made to use POSIX calls throughout.</td>
<td>0 There are some machine dependent data representations</td>
</tr>
<tr>
<td>0 System dependencies platform or COTS are encapsulated.</td>
<td>0 Code is not ANSI standard</td>
</tr>
<tr>
<td>0 No documented method for other languages to call services</td>
<td>0 Variables used for other than their declared purpose</td>
</tr>
<tr>
<td>0 “Man pages” are out of date for some APIs</td>
<td>0 No low-level control and task flows in documentation</td>
</tr>
<tr>
<td>0 Number of modules may be excessive</td>
<td>0 No prologs for the majority of the modules</td>
</tr>
<tr>
<td>0 COTS screen description files use standard X-Windows resource file formats</td>
<td>0 Inadequate indexing of documentation</td>
</tr>
<tr>
<td>0 Proprietary language does not support data typing</td>
<td>0 Excessive use of global variables</td>
</tr>
<tr>
<td>0 In the vendor’s proprietary language, variables are never explicitly declared (A typo will create a variable)</td>
<td>0 Input error checking is not consistently applied</td>
</tr>
<tr>
<td>0 SQL is only used for ~10% of the code that accesses the database</td>
<td>0 System dependent on a proprietary language for some functions related to integration with COTS</td>
</tr>
<tr>
<td>- The rest uses the proprietary DBMS calls</td>
<td>0 Lack of consistency in the code between programmers</td>
</tr>
<tr>
<td>0 Complete source code for gnu Perl was included as part of deliverable subsystem source code</td>
<td>0 No isolation or encapsulation of dependencies on platform or COTS</td>
</tr>
<tr>
<td>0 System tied to a proprietary language for procedural processing and data access</td>
<td>0 System is dependent on a proprietary run-time environment</td>
</tr>
<tr>
<td>0 System is dependent on a proprietary run-time environment</td>
<td>0 Fourteen violations of one of the few company coding standards</td>
</tr>
<tr>
<td>0 Two percent of the code modules are overly large, more than 100 LOC</td>
<td>0 Two percent of the code modules are overly large, more than 100 LOC</td>
</tr>
</tbody>
</table>

---

This Chart Contains Representative Assessment Results
Examples of Software Quality Risk Profiles (3D)
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 req’ts
  - 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
Discussion Outline

0 Introduction
   - Managing S/W Quality Issues by Measuring Risks
0 Background
   - S/W Quality Risks
   - Metrics and Measures
0 Discussion
   - Components of a SW Risk Management Framework
   - Core SW Risks vs. Technology and Project Choices
0 Summary
   - Using SW Risk Assessments to Manage
   - Transferring to Industry and Academia
   - Adapting SQAE to…
Direct and Indirect Technology Transfers of SQAE Methods and Tools

SQAE™ Licensees:
- Mitretek
- Lockheed Martin
- Grumman Data Systems
- General Dynamics Info. Systems
- TRW Systems & Info. Tech. Group

Organizations Who’s SW was Assessed:
- 1996: Mitretek
- 1997: Lockheed Martin, Grumman Data Systems
- 1999: Ecole de technologie superieure
- 2001: SEI

Presentations:
- IEEE CS May 1994
- 4th Int’l SW Quality Conf. April 1995
- STSC May 1996
- SEI SEPG May 1996
- STSC April 1998
- STSC May 2002
- OSD/SEI Jan 2003
- BCS SQM Apr 2004

--> ISO 9126-1
--> Arch/Design Quality
--> Secureness

MITRE
SQAE Experience-Base

MITRE
Our Collaboration Partners are Gaining Experience Too…

La méthode SQAE en bref

Ces caractéristiques de qualité sont fondées sur 7 facteurs de qualité, eux-mêmes comportant plusieurs attributs de qualité. Ces facteurs de qualité sont:

- Cohérence (comporte 15 attributs);
- Indépendance (comporte 8 attributs);
- Modularité (comporte 10 attributs);
- Documentation (comporte 16 attributs);
- Auto-documentation;
- Contrôle d’anomalie;
- Simplification.

Pointage des domaines de qualité

<table>
<thead>
<tr>
<th>Zone de qualité</th>
<th>Résultat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maintenabilité</td>
<td>80</td>
</tr>
<tr>
<td>Evolubilité</td>
<td>77</td>
</tr>
<tr>
<td>Portabilité</td>
<td>78</td>
</tr>
<tr>
<td>Descriptivité</td>
<td>76</td>
</tr>
</tbody>
</table>

Pointage des facteurs de qualité

<table>
<thead>
<tr>
<th>Référence</th>
<th>Facteur de qualité</th>
<th>Résultat</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Cohérence</td>
<td>100</td>
</tr>
<tr>
<td>2</td>
<td>Indépendance</td>
<td>83</td>
</tr>
<tr>
<td>3</td>
<td>Modularité</td>
<td>72</td>
</tr>
<tr>
<td>4</td>
<td>Documentation</td>
<td>76</td>
</tr>
<tr>
<td>5</td>
<td>Auto-descriptivité</td>
<td>76</td>
</tr>
<tr>
<td>6</td>
<td>Contrôle d’anomalie</td>
<td>78</td>
</tr>
<tr>
<td>7</td>
<td>Simplicité du design</td>
<td>82</td>
</tr>
</tbody>
</table>

Évolubilité

- Zone de risque faible
- Zone de risque modéré
- Zone de risque élevé
Access to SQAE Information...

Software Quality Assessment Exercise (SQAE)

As software undergoes maintenance and enhancement, it becomes brittle, complex, and more susceptible to errors. Software quality teams cannot afford to focus simply on removing errors; the fundamental software architectural issues of evolvability, portability, and maintainability are key to an organization's continued ability to survive and thrive.

MITRE has developed a Software Quality Assessment Exercise (SQAE) methodology and framework specifically to provide insights about software quality from a comprehensive, life-cycle engineering perspective. The SQAE methodology is designed to be tailored to the specific context and needs of a particular organization while maintaining a comprehensive perspective on the risk areas of software-based systems. The SQAE capitalizes on industry and government open system standards, on MITRE's past efforts and experience in conducting over 100 assessments in this area, and on modern software reverse-engineering analysis tools and code assessment technologies. The SQAE has been applied to more than 60 languages and a total of over 50 million lines of code.

MITRE's software quality assessment methodology incorporates a framework of measures for investigating the life-cycle quality risks associated with software systems, in both the code itself and its supporting documentation. The breadth and depth of the framework's structure ensures that each assessment is performed to the same level of thoroughness, and that the results point an accurate and meaningful portrait of the risk drivers and risk mitigators present in the system. The SQAE also factors in any project-specific information that can be used to guide the assessment and lead to a tighter focus in the discussion of risks. One of the key tenets of the software quality assessment framework is that it be flexible enough to allow for the analysis of systems in any language, on any platform, whether they use commercial packages or not. The methodology's design makes no assumptions regarding documentation or implementation details, and can be applied to varying architectures and development methods.

Management is provided with clear and objective information about the risks associated with their application systems. Taken together, the risk drivers and risk mitigators identified through the assessment clearly depict a software-centric profile of the risks associated with each software product developed by an organization. This profile can be used to guide migration decisions, identify needed changes in an organization's development methods, and guide management's software quality actions during the evolution of the organization's application systems.

For Information About Obtaining a No-Cost License for the SQAE

A license for this tool can be obtained through the MITRE Technology Transfer Office. For More Information About the SQAE For more information about the SQAE please make use of the following resources:


http://www.mitre.org/worktech_transfer/sqae.html
Discussion Outline

0 Introduction
   - Managing S/W Quality Issues by Measuring Risks

0 Background
   - S/W Quality Risks
   - Metrics and Measures

0 Discussion
   - Components of a SW Risk Management Framework
   - Core SW Risks vs. Technology and Project Choices

0 Summary
   - Using SW Risk Assessments to Manage
   - Transferring to Industry and Academia
   - Adapting SQAE to…
Collaborating on the Enhancement of SQAE

- Assess secureness
- Assess maintainability
- Assess descriptiveness
- Assess portability
- Assess evolvability

64 legacy and hierarchical languages of original SQAE™

Model-based development

- Event-driven programming
- Component-based development
- Development in .NET or J2EE Frameworks
- Web-services development

ISO/IEC 9126, model of quality characteristics
ISO/IEC 14598, product quality evaluation processes
IEEE/EIA 12207, life cycle process standard family
- IEEE Std 1540 risk management process
- ISO/IEC 15939 measurement process
Reworking the SQAE Measures with ISO/IEC 9126-1’s Metrics
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.

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

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

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

For non-OO design/implementations

3.1 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.

3.2 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.

3.10 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.

For OO design/implementations

3.1 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 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 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.

<table>
<thead>
<tr>
<th>Activity</th>
<th>Description</th>
<th>Formula/Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.2</td>
<td>Is the source code of low complexity (e.g., McCabe Cyclomatic…)?</td>
<td>Using the available source code, calculate the McCabe Cyclomatic complexity.</td>
</tr>
<tr>
<td>7.7</td>
<td>Is the code segmented into procedure bodies that can be understood easily?</td>
<td>Using the available source code, examine the code.</td>
</tr>
<tr>
<td>7.9</td>
<td>Have all procedures been structured to avoid excessive nesting?</td>
<td>Using the available source code, calculate the average and standard deviation of the nesting levels.</td>
</tr>
<tr>
<td>For non-OO design/implementations</td>
<td></td>
<td>For OO design/implementations</td>
</tr>
<tr>
<td>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.</td>
<td>Rating will be between 1 and 0, where 1 is the higher rating. Score will be [1 - \left(\frac{\text{average} + \text{s.d.}}{15}\right) \times 0.02].</td>
<td></td>
</tr>
</tbody>
</table>

**WMC**
Weighted Methods Per Class

**NOC**
Number of Children

**DIT**
Depth of Inheritance Tree of a Class

**Notes**
- For non-OO design/implementations
- For OO design/implementations

**MITRE**
Robert A. Martin
781-271-3001
ramartin@mitre.org