Managing Software Risks in Software Intensive Systems with Metrics and Measures

Robert A. Martin
30 January 2003
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
  - Adapting to Handle OO Specific Risks

0 Summary
  - Using SW Risk Assessments to Manage
  - Transferring to Industry and Academia
If, To Manage You Must Measure… How Do You Measure an Abstract Concept Like Software Risk?
One Method of Assessing Software Risks
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.
Introduction:
MITRE’s Software Risk Assessment Work

- Wanted a framework for assessing lifecycle quality issues
- 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
- 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
- MITRE’s 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

0 Discussion
    - Components of a SW Risk Management Framework
    - Adapting to Handle OO Specific Risks

0 Summary
    - Using SW Risk Assessments to Manage
    - Transferring to Industry and Academia
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.

M O T I F
POSIX
ANSI SQL

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

NIFTY HACKS
of the SUN GURUS

Secrets of the file system

MITRE
Establishing a Framework for Measuring Risks

0 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
0 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
0 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
- Rome Laboratory -- QUES
- Rome Laboratory -- SQT2 -- Software Quality Technology Transfer Consortium
- ESPRIT REQUEST Kitchenham et al 1987
- ESPRIT REQUEST Kitchenham et al 1989
- Computer Technology
  - Processing Power
  - Memory Sizes
  - Disk Sizes
- Assessment Technology
- USAF SSG SEP code review
- AFOTEC Software Assessment Team Pamphlets
- 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
   - Adapting to Handle OO Specific Risks

0 Summary
   - Using SW Risk Assessments to Manage
   - Transferring to Industry and Academia
Targeted Attributes Of Our S/W Quality Risk Assessment Methodology

The assessment should be:
- 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
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
0 Discussion
  - Components of a SW Risk Management Framework
  - Adapting to Handle OO Specific Risks
0 Summary
  - Using SW Risk Assessments to Manage
  - Transferring to Industry and Academia
Bridging the Gap between The Measurable and Unmeasurable

Software Quality Analysis Frameworks

Maintainability
Evolvability
Portability
Descriptiveness

Complexity
Set-Use
SLOC Count
API Utilization
Coding Standards

Quality Concepts

Metrics & Measures

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

Boehm et. al. RADC
Original Quality Factors 1977

Measurable Property
- device independence
- completeness
- accuracy
- consistency
- accessibility
- communicativeness
- self-descriptiveness
- conciseness
- legibility
- augmentability

Perceived Aspect of Quality
- portability
- reliability
- efficiency
- human engineering
- testability
- understandability
- modifiability

McCall et. al. RADC
Modified Quality Model 1984

Measurable Property
- maintainability
- consistency
- accuracy
- error tolerance
- execution efficiency
- storage efficiency
- access control
- access audit
- operability
- training
- communicativeness
- simplicity
- self-descriptiveness
- expandability
- modularity
- software system independence
- machine independence
- communications commonality

Perceived Aspect of Quality
- testability
- analyzability
- comprehensiveness
- integrity
- re-usability
- testability
- expandability

MITRE SQAE ™
Quality Factors 1992 - 2003

Measurable Property
- maintainability
- consistency
- accuracy
- error tolerance
- execution efficiency
- storage efficiency
- access control
- access audit
- operability
- training
- communicativeness
- simplicity
- self-descriptiveness
- expandability
- modularity
- software system independence
- machine independence
- communications commonality

Perceived Aspect of Quality
- testability
- analyzability
- comprehensiveness
- integrity
- re-usability
- testability
- expandability

AFOTEC Supportability Assessment Factors 1991 - 1996

Measurable Property
- maintainability
- consistency
- accuracy
- error tolerance
- execution efficiency
- storage efficiency
- access control
- access audit
- operability
- training
- communicativeness
- simplicity
- self-descriptiveness
- expandability
- modularity
- convention

- requirements issues
- testing issues
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

- software attributes

1992 - 2003
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.

Components of the Software Quality Areas

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

Specific Questions about the Code and Documentation

Source Code and Listings

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

Quality Risk Profile: Risk Mitigators
1. Coding conventions, style and indentation help ensure code is readable
2. Software to be ported to other environments
3. Unit tests to ensure code is portable
4. Source code documentation and interfaces
5. Portability testing of software to be ported
6. Source code tracking and version control
7. Integration testing and code reviews
8. Documentation and user manuals

Quality Risk Profile: Risk Drivers
1. Software being developed for multiple platforms
2. Lack of documentation and testing
3. Complex software interfaces
4. Lack of unit testing
5. Inconsistent code styling
6. Lack of documentation

Software Quality Risk Profiles

- Portability
- Evolvability
- Maintainability
- Descriptiveness

MITRE
Putting the Pieces Together

Maintainability

- Design
- Simplicity
- Consistency
- Modularity
- Self-Descriptiveness
- Anomaly Control
- Documentation

Evolvability

- Design
- Simplicity
- Modularity
- Anomaly Control
- Documentation
- Self-Descriptiveness

Portability

- Modularity
- Independence
- Self-Descriptiveness
- Documentation

“Documentation”

- Documentation
- Self-Descriptiveness

“Modularity”

- Modularity
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.

<table>
<thead>
<tr>
<th>Proportion of Factor Value</th>
</tr>
</thead>
</table>

**Software System Independence**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Does the software avoid all usage of specific pathnames/filenames? (100)</td>
</tr>
<tr>
<td>2.2</td>
<td>Is the software free of machine, OS and vendor specific extensions? (200)</td>
</tr>
<tr>
<td>2.3</td>
<td>Are system dependent functions, etc., in stand-alone modules (not embedded in the code)? (200)</td>
</tr>
<tr>
<td>2.4</td>
<td>Are the languages and interface libraries selected standardized and portable? (i.e., ANSI…) (200)</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)? (100)</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? (100)</td>
</tr>
</tbody>
</table>

**Machine Independence**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7</td>
<td>Is the data representation machine independent? (200)</td>
</tr>
<tr>
<td>2.8</td>
<td>Are the commercial software components available on other platforms in the same level of functionality? (200)</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.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></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? (200)</td>
</tr>
<tr>
<td>3.2</td>
<td>Do the functional groupings of units avoid calling units outside their functional area? (150)</td>
</tr>
<tr>
<td>3.3</td>
<td>Are machine dependent and I/O functions isolated and encapsulated? (150)</td>
</tr>
<tr>
<td>3.4</td>
<td>Are interpreted code bodies (shell scripts and 4GL scripts) protected from accidental or deliberate modification? (150)</td>
</tr>
<tr>
<td>3.5</td>
<td>Do all functional procedures represent one function (one-to-one function mapping)? (150)</td>
</tr>
<tr>
<td>3.6</td>
<td>Are all commercial software interfaces &amp; APIs, other than GUI Builders, isolated and encapsulated? (200)</td>
</tr>
<tr>
<td>3.7</td>
<td>Have symbolic constants been used in place of explicit ones? (150)</td>
</tr>
<tr>
<td>3.8</td>
<td>Are symbolic constants defined in an isolated and centralized area? (100)</td>
</tr>
<tr>
<td>3.9</td>
<td>Are all variables used exclusively for their declared purposes? (150)</td>
</tr>
<tr>
<td>3.10</td>
<td>Has the code been structured to minimize coupling to global variables? (100)</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 standards maintained during software development. Consistency in a code that forms a standard document and development is carried out in conformance with this document throughout. Any potential issues on documentation, I/O protocols, data definition and nomenclature, etc., are represented here:

1. Is there a representation of the design in the paper documentation?
2. Are there consistent language, unit, and data type definitions?
3. Is there a standard format of documentation? (e.g., ADA, SQL, Ada, C, Fortran, etc.)
4. Are naming standards consistent across (OSLCs)?
5. Is there a standard for function naming in the paper documentation?
6. Are naming conventions consistent for functional groupings?
7. Are naming conventions consistent for usage (e.g., I/O, constants, etc.)?
8. Does the paper documentation establish accuracy requirements for computations?
9. Are there quantitative accuracy requirements stated in the paper documentation for all constants?
10. Are the naming conventions consistent for functional groupings?
11. Are the naming conventions consistent for usage (e.g., I/O, constants, etc.)?
12. Are the naming conventions consistent for data type (e.g., constant boolean), etc.?

**Independence**
Independence refers to two broad groups: software system independence and machine independence. The issue is to not be the system to any specific host environment which would make it difficult or impossible to migrate, evolve, or enhance the system.

1. Is the software free of machine, OS and vendor specific extensions?
2. Are system dependent functions, etc., in stand-alone modules (not embedded in the code)?
3. Do the languages and interface libraries selected standardized and portable? (e.g., ANSI...)
4. Does the software avoid the need for any unique compilation in order to run (e.g., a custom post processor to "tweak"

**Modularity**
Modularity consists of several facets which each support the concepts of organized separation of functions and maximize the robust coupling between portions of the system.

1. Is the structure of the design hierarchical in a top-down design within tasking threads?
2. Is the design documentation display control flow in the CSS/CSC level?
3. Is the design documentation display data flow?
4. Is the design documentation depict the tasks and system initialization hierarchy/relationship?
5. Are the documentation adequately illustrated (functionality can be easily located in a single view)?
6. Does the documentation contain comprehensive descriptions of all software functionality, software interfaces, algorithms, commercial components, system software functions, interprocess communications, and interfaces?
7. Is the paper documentation elaborate a requirement for communicating global data within a software unit to show where the data is defined, the data composition and how the data is used?

**Documentation**
Documentation refers to the external printed material of the system. The concern is that the documentation be adequate to support the maintenance, porting, and enhancement activities which will occur throughout the systems life.

1. Is the documentation of the design in the paper documentation?
2. Read paper documentation and verify the presence of the information.
3. Rating will be between 1 and 0, where 1 is the higher rating.
4. Is there a definition of standard I/O handling in the paper documentation?
5. Read paper documentation and verify the presence of the information.
6. Rating will be between 1 and 0, where 1 is the higher rating.

**Self-Descriptiveness**
Self-descriptiveness makes it easy to understand the purpose and use of each module and its data flow and interaction with other modules. It reduces the learning curve for new developers and assists in maintaining and testing the system.

1. Are the naming conventions consistent for usage (e.g., I/O, constants, etc.)?
2. Are the naming conventions consistent for data type (e.g., constant boolean), etc.?

**Anomaly Control**
Design Simplicity

1. Are all inputs, process and outputs adequately defined in the documentation?
2. Read paper documentation and verify the presence of the information for the modules used in 1.11.
3. Rating will be between 1 and 0, where 1 is the higher rating. 1 for a good design, 0 if consistent but no standard defined, 0.0 if neither.

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

1. Are the naming conventions consistent for functional groupings?
2. Are the naming conventions consistent for usage (e.g., I/O, constants, etc.)?
3. Does the paper documentation establish accuracy requirements for computations?
4. Are there quantitative accuracy requirements stated in the paper documentation for all constants?
5. Are the naming conventions consistent for usage (e.g., I/O, constants, etc.)?
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.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1.10</strong></td>
<td>Are the naming conventions consistent for functional groupings?</td>
</tr>
<tr>
<td></td>
<td>Examine the scheduling modules and one other large functional grouping and cross reference between them.</td>
</tr>
<tr>
<td></td>
<td>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><strong>2.2</strong></td>
<td>Is the software free of machine, OS and vendor specific extensions?</td>
</tr>
<tr>
<td></td>
<td>Examine two large functional groupings of code and cross reference between them and system libraries and known vendor extensions.</td>
</tr>
<tr>
<td></td>
<td>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><strong>2.3</strong></td>
<td>Are system dependent functions, etc., in stand-alone modules (not embedded in the code)?</td>
</tr>
<tr>
<td></td>
<td>Examine all known instantiations OS and vendor specific dependencies for encapsulation/isolation.</td>
</tr>
<tr>
<td></td>
<td>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>
The Core of the Software Quality Assessment Methodology

1. Identify risks to be evaluated.
2. Determine what is needed to evaluate the risk.
3. Determine whether the needed items are available; if items are not obtainable then discard that risk from the scope of the assessment.
4. Determine how the items will be examined to determine the risk information.
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
   - Adapting to Handle OO Specific Risks
0 Summary
   - Using SW Risk Assessments to Manage
   - Transferring to Industry and Academia
# 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.

<table>
<thead>
<tr>
<th>Activity</th>
<th>Description</th>
<th>Rating Criteria</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? Has the code been structured into cohesive classes?</td>
<td>Using the two large functional areas of the code from 1.10, run McCabe BattleMap, Refine, or another tool to produce a structure chart. Examine. Rating will be between 1 and 0, where 1 is the higher rating. 1 for crisp hierarchy, 0.6 for discernible hierarchy, 0 if neither.</td>
</tr>
<tr>
<td>3.2</td>
<td>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?</td>
<td>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, 0.6 for generally isolated, 0 if neither.</td>
</tr>
<tr>
<td>3.10</td>
<td>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?</td>
<td>Examine the modules from 1.10, 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 the classes in these modules coupled to other classes to total classes in the system.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**For non-OO design/implementations**

**For OO design/implementations**

<table>
<thead>
<tr>
<th>Activity</th>
<th>Description</th>
<th>Rating Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## 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>For non-OO design/implementations</th>
<th>For OO design/implementations</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.2 Is the source code of low complexity (e.g., McCabe Cyclomatic…)?</td>
<td>Using the available source code, calculate the Cyclomatic complexity.</td>
</tr>
<tr>
<td>Are the classes of low complexity?</td>
<td>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].</td>
</tr>
<tr>
<td><strong>WMC</strong> Weighted Methods Per Class</td>
<td></td>
</tr>
<tr>
<td>7.7 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>Has code been organized into classes and sub-classes that can be understood easily?</td>
<td>Rating will be between 1 and 0, where 1 is the higher rating. Score will be 1 - [(average + s.d. - 15) x 0.02].</td>
</tr>
<tr>
<td><strong>NOC</strong> Number of Children</td>
<td></td>
</tr>
<tr>
<td>7.9 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>Have classes been structured to avoid large levels of inheritance?</td>
<td>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.</td>
</tr>
<tr>
<td><strong>DIT</strong> Depth of Inheritance Tree of a Class</td>
<td></td>
</tr>
</tbody>
</table>

---

For non-OO design/implementations:

- **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].

- **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. Score will be 1 - [(average + s.d. - 15) x 0.02].

- **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.

For OO design/implementations:

- **WMC** (Weighted Methods Per Class): 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].

- **NOC** (Number of Children): 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. Score will be 1 - 20 * the average ratio of classes to the direct sub-classes.

- **DIT** (Depth of Inheritance Tree of a Class): 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

0 Discussion
   - Components of a SW Risk Management Framework
   - Adapting to Handle OO Specific Risks

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

The Selection of Contractors

Reviews of SIW Releases for a Project Office

Selection of Migration Systems

MITRE
Software Quality Assessment Uses

0 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
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™ Foundation

<table>
<thead>
<tr>
<th>Source Code</th>
<th>Project A</th>
<th>Project B</th>
<th>...</th>
<th>Project CZ</th>
<th>Total of Projects</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>112,000 LOC</td>
<td>558,000 LOC</td>
<td></td>
<td>58,000 LOC</td>
<td>51,173,315 LOC</td>
</tr>
<tr>
<td></td>
<td>Ada, C, Shell, TAE+, SQL, X, MOTIF</td>
<td>C, Shell, X, MOTIF</td>
<td>Ada, C, ELF, ezX, SQL, X, MOTIF</td>
<td>Ada, C, FORTRAN, COBOL, shell, TAE+, SQL, X, MOTIF, UIL, Stored Procedures, GEL, ELF, ezX, ...</td>
<td></td>
</tr>
<tr>
<td>Written Material</td>
<td>Top Level Design Doc</td>
<td>SDD</td>
<td>SDD</td>
<td>Top Level Design Doc</td>
<td>Top Level Design Doc</td>
</tr>
<tr>
<td>Reference Material</td>
<td>SDD</td>
<td>SDD</td>
<td>SPS SDD SDP</td>
<td>SPS SDD SDP</td>
<td>SPS SDD SDP Case Tools Repositories</td>
</tr>
<tr>
<td>COTS Manuals &amp; Articles</td>
<td>Product Literature</td>
<td>Design and Code Stnds</td>
<td>Product Literature</td>
<td>Design and Code Stnds</td>
<td></td>
</tr>
</tbody>
</table>

This Chart Contains Representative Assessment Results
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 design documentation, showing data and control flows.</td>
<td>0 Lack of standards for naming conventions, error handling data definitions, I/O, etc</td>
</tr>
<tr>
<td>0 Good developer documentation for supported APIs.</td>
<td>0 Design documentation is poorly organized, incomplete, and at a very high level</td>
</tr>
<tr>
<td>0 Good top-down hierarchical structure to code.</td>
<td>0 No low-level design information or functional allocation of software in documentation</td>
</tr>
<tr>
<td>0 Modules use straightforward algorithms in a linear fashion.</td>
<td>0 Machine generated code documentation is inconsistent with the developed code documentation</td>
</tr>
<tr>
<td>0 System dependencies are to readily available COTS software.</td>
<td>0 Machine generated code is undocumented</td>
</tr>
<tr>
<td>0 Code is of low complexity.</td>
<td>0 Procedure and file names depend on path for uniqueness</td>
</tr>
<tr>
<td>0 Logic flow through individual procedures is easy to follow.</td>
<td>0 Hard coded absolute filenames/paths used</td>
</tr>
<tr>
<td>0 Disciplined coding standards followed by the programmers.</td>
<td>0 UNIX commands hardcoded in the code</td>
</tr>
<tr>
<td>0 Considerable effort made to use POSIX calls throughout.</td>
<td>0 Hard coded variables used when symbolic constants should have been used</td>
</tr>
<tr>
<td>0 System dependencies platform or COTS are encapsulated.</td>
<td>0 There are some machine dependent data representations</td>
</tr>
<tr>
<td></td>
<td>0 Code is not ANSI standard</td>
</tr>
<tr>
<td></td>
<td>0 Variables used for other than their declared purpose</td>
</tr>
<tr>
<td></td>
<td>0 No low-level control and task flows in documentation</td>
</tr>
<tr>
<td></td>
<td>0 No prologs for the majority of the modules</td>
</tr>
<tr>
<td></td>
<td>0 Inadequate indexing of documentation</td>
</tr>
<tr>
<td></td>
<td>0 Excessive use of global variables</td>
</tr>
<tr>
<td></td>
<td>0 Input error checking is not consistently applied</td>
</tr>
<tr>
<td></td>
<td>0 System dependent on a proprietary language for some functions related to integration with COTS</td>
</tr>
<tr>
<td></td>
<td>0 Lack of consistency in the code between programmers</td>
</tr>
<tr>
<td></td>
<td>0 No isolation or encapsulation of dependencies on platform or COTS</td>
</tr>
<tr>
<td></td>
<td>0 System tied to a proprietary language for procedural processing and data access</td>
</tr>
<tr>
<td></td>
<td>0 System is dependent on a proprietary run-time environment</td>
</tr>
<tr>
<td></td>
<td>0 Fourteen violations of one of the few company coding standards</td>
</tr>
<tr>
<td></td>
<td>0 Two percent of the code modules are overly large, more than 100 LOC</td>
</tr>
</tbody>
</table>

Other Observations

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

This Chart Contains Representative Assessment Results

MITRE
Examples of Feedback

Application’s Primary Strengths: Integrator Perspective

0 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 OS, new COTS Database, etc) code rewrites and unit tests should be restricted to distinct areas rather than rippling throughout.

Application’s Primary Weaknesses: Integrator Perspective

0 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.
Sample Assessment Results for Multiple Developers

Maintainability | Evolvability | Portability | Descriptiveness

Quality Area Risk Profiles

Software Quality Factor Risk Profiles

Consistency | Independence | Modularity

Documentation | Self-Descriptiveness | Anomaly Control | Design Simplicity

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
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
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
   - Adapting to Handle OO Specific Risks
0 Summary
   - Using SW Risk Assessments to Manage
   - Transferring to Industry and Academia
Direct and Indirect Technology Transfers of SQAE™ Methods and Tools

SQAE™ Licensees:
- 1996 - Mitretek
- 1997 - Lockheed Martin - Grumman Data Systems
- 2001 - Ecole de technologie superieure

Organizations Who’s SW was Assessed:
- 1993
- 1994
- 1995
- 1996
- 1997
- 1998/1999
- 2000
- 2001
- 2002
- 2003

IEEE CS May 1994
Managing Software Quality Throughout the Lifecycle
STSC April 1995
4th Int’l SW Quality Conf. July 1994

INCOSE July 1996
Presence a Framework for Effective Risk Management
STSC May 2002

IEEE CS May 1994
STSC April 1995
SEI SEPG May 1996
STSC April 1998

MITRE
Languages from Product Assessments By Percentage of Assessed LOC

100 Systems with approximately 51 Million Lines of Code Assessed
Why Are We Interested In S/W Risk Anyway?

Software Quality Teams must do more than remove errors!

Constrained Resources

New Requirements and Needs

C³I Systems

New Technology

Changing Operational Environment

change

MITRE
So, the SQAE is Only One Example of the Risk Assessments that use Metrics and Measures
Metadata and Data Handling

Quality Factors

- Consistency
  - LDM-PDM Congruence
  - Standards Use
  - Data Integrity
  - GUI Code

- Independence
  - Standards Use
  - Data Model
  - Standards Use

- Design Simplicity
  - OS/Machine Dependencies
  - Customizations
  - GUI Code Encapsulation

- Modularity
  - Platform Availability
  - Data Model
  - Encapsulation

- Anomaly Control
  - Run-time Issues
  - Consistent Error Handling
  - 4GL Design Tools

- Documentation
  - 3rd Party Tools
  - 4GL Use
  - GUI Code Isolation

- Self-Descriptiveness
  - Performance Management
  - DBMS Design Tools

Quality of Metadata and Data Handling

Maintainability
Evolvability
Portability
Descriptiveness
A “Design Review-time” View of the SQAE

- Independence
- Design Simplicity
- Modularity
- Consistency
- Anomaly Control
- Documentation
- Self Descriptiveness

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

A-SPEC

SDP

Detailed Design

SQAE Quality Factors
Augmented SCE: Implementation Schedule/Activities

Summary
Initial Planning, Detailed Preparation Work, IFPP Material Development:
Assessment Activities per contractor (Process team visits cover 3 projects, Product team evaluates 2 of those 3 projects):
Report Generation and Briefing Creation/Delivery

- 4 staff weeks
- 7 staff weeks per contractor
- 10 staff weeks