Quality Management

- Managing the quality of the software process and products

Objectives

- To introduce the quality management process and key quality management activities
- To explain the role of standards in quality management
- To explain the concept of a software metric, predictor metrics and control metrics
- To explain how measurement may be used in assessing software quality

Topics covered

- Quality assurance and standards
- Quality planning
- Quality control
- Software measurement and metrics

Software quality management

- Concerned with ensuring that the required level of quality is achieved in a software product
- Involves defining appropriate quality standards and procedures and ensuring that these are followed
- Should aim to develop a ‘quality culture’ where quality is seen as everyone’s responsibility
What is quality?
• Quality, simplistically, means that a product should meet its specification
• This is problematical for software systems
  – Tension between customer quality requirements (efficiency, reliability, etc.) and developer quality requirements (maintainability, reusability, etc.)
  – Some quality requirements are difficult to specify in an unambiguous way
  – Software specifications are usually incomplete and often inconsistent

The quality compromise
• We cannot wait for specifications to improve before paying attention to quality management
• Must put procedures into place to improve quality in spite of imperfect specification
• Quality management is therefore not just concerned with reducing defects but also with other product qualities

Quality management activities
• Quality assurance
  – Establish organisational procedures and standards for quality
• Quality planning
  – Select applicable procedures and standards for a particular project and modify these as required
• Quality control
  – Ensure that procedures and standards are followed by the software development team
• Quality management should be separate from project management to ensure independence

ISO 9000
• International set of standards for quality management
• Applicable to a range of organisations from manufacturing to service industries
• ISO 9001 applicable to organisations which design, develop and maintain products
• ISO 9001 is a generic model of the quality process
• Must be instantiated for each organisation

ISO 9001
<table>
<thead>
<tr>
<th>Management responsibility</th>
<th>Quality system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control of non-conforming products</td>
<td>Design control</td>
</tr>
<tr>
<td>Handling, storage, packaging and delivery</td>
<td>Purchasing</td>
</tr>
<tr>
<td>Purchaser-supplied products</td>
<td>Product identification and traceability</td>
</tr>
<tr>
<td>Process control</td>
<td>Inspection and testing</td>
</tr>
<tr>
<td>Inspection and test equipment</td>
<td>Inspection and test status</td>
</tr>
<tr>
<td>Contract review</td>
<td>Corrective action</td>
</tr>
<tr>
<td>Document control</td>
<td>Quality records</td>
</tr>
<tr>
<td>Internal quality audits</td>
<td>Training</td>
</tr>
<tr>
<td>Servicing</td>
<td>Statistical techniques</td>
</tr>
</tbody>
</table>
ISO 9000 certification

- Quality standards and procedures should be documented in an organisational quality manual.
- An external body may certify that an organisation’s quality manual conforms to ISO 9000 standards.
- Customers are increasingly demanding that suppliers are ISO 9000 certified.

ISO 9000 and quality management

- Standards are the key to effective quality management.
- They may be international, national, organizational or project standards.
- Product standards define characteristics that all components should exhibit e.g. a common programming style.
- Process standards define how the software process should be enacted.

Quality assurance and standards

- Standards are the key to effective quality management.
- They may be international, national, organizational or project standards.
- Product standards define characteristics that all components should exhibit e.g. a common programming style.
- Process standards define how the software process should be enacted.

Importance of standards

- Encapsulation of best practice - avoids repetition of past mistakes.
- Framework for quality assurance process - it involves checking standard compliance.
- Provide continuity - new staff can understand the organisation by understanding the standards applied.

Product and process standards

<table>
<thead>
<tr>
<th>Product standards</th>
<th>Process standards</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design review form</td>
<td>Design review conduct</td>
</tr>
<tr>
<td>Document naming standards</td>
<td>Submission of documents to CM</td>
</tr>
<tr>
<td>Procedure header format</td>
<td>Version release process</td>
</tr>
<tr>
<td>Ada programming style standard</td>
<td>Project plan approval process</td>
</tr>
<tr>
<td>Project plan format</td>
<td>Change control process</td>
</tr>
<tr>
<td>Change request form</td>
<td>Test recording process</td>
</tr>
</tbody>
</table>

Problems with standards

- Not seen as relevant and up-to-date by software engineers.
- Involve too much bureaucratic form filling.
- Unsupported by software tools so tedious manual work is involved to maintain standards.
Standards development

- Involve practitioners in development. Engineers should understand the rationale underlying a standard.
- Review standards and their usage regularly. Standards can quickly become outdated and this reduces their credibility amongst practitioners.
- Detailed standards should have associated tool support. Excessive clerical work is the most significant complaint against standards.

Documentation standards

- Particularly important - documents are the tangible manifestation of the software.
- Documentation process standards
  - How documents should be developed, validated, and maintained.
- Document standards
  - Concerned with document contents, structure, and appearance.
- Document interchange standards
  - How documents are stored and interchanged between different documentation systems.

Documentation process

- Document identification standards
  - How documents are uniquely identified.
- Document structure standards
  - Standard structure for project documents.
- Document presentation standards
  - Define fonts and styles, use of logos, etc.
- Document update standards
  - Define how changes from previous versions are reflected in a document.

Document interchange standards

- Documents are produced using different systems and on different computers.
- Interchange standards allow electronic documents to be exchanged, mailed, etc.
- Need for archival. The lifetime of word processing systems may be much less than the lifetime of the software being documented.
- XML is an emerging standard for document interchange which will be widely supported in future.

Process and product quality

- The quality of a developed product is influenced by the quality of the production process.
- Particularly important in software development as some product quality attributes are hard to assess.
- However, there is a very complex and poorly understood between software processes and product quality.
Process-based quality

- Straightforward link between process and product in manufactured goods
- More complex for software because:
  - The application of individual skills and experience is particularly important in software development
  - External factors such as the novelty of an application or the need for an accelerated development schedule may impair product quality
- Care must be taken not to impose inappropriate process standards

Practical process quality

- Define process standards such as how reviews should be conducted, configuration management, etc.
- Monitor the development process to ensure that standards are being followed
- Report on the process to project management and software procurer

Quality planning

- A quality plan sets out the desired product qualities and how these are assessed and define the most significant quality attributes
- It should define the quality assessment process
- It should set out which organisational standards should be applied and, if necessary, define new standards

Quality plan structure

- Product introduction
- Product plans
- Process descriptions
- Quality goals
- Risks and risk management
- Quality plans should be short, succinct documents
  - If they are too long, no-one will read them

Software quality attributes

<table>
<thead>
<tr>
<th>Quality attribute</th>
<th>Qualifier 1</th>
<th>Qualifier 2</th>
<th>Qualifier 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Safety</td>
<td>Understandability</td>
<td>Portability</td>
<td>Security</td>
</tr>
<tr>
<td>Security</td>
<td>Reliability</td>
<td>Adaptability</td>
<td>Reusability</td>
</tr>
<tr>
<td>Reliability</td>
<td>Resilience</td>
<td>Modularity</td>
<td>Efficiency</td>
</tr>
<tr>
<td>Resilience</td>
<td>Robustness</td>
<td>Complexity</td>
<td>Learnability</td>
</tr>
</tbody>
</table>
Quality control

• Checking the software development process to ensure that procedures and standards are being followed
• Two approaches to quality control
  – Quality reviews
  – Automated software assessment and software measurement

Quality reviews

• The principal method of validating the quality of a process or of a product
• Group examined part or all of a process or system and its documentation to find potential problems
• There are different types of review with different objectives
  – Inspections for defect removal (product)
  – Reviews for progress assessment (product and process)
  – Quality reviews (product and standards)

Types of review

<table>
<thead>
<tr>
<th>Review type</th>
<th>Principal purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design or program inspections</td>
<td>To detect detailed errors in the design or code and to check whether standards have been followed. The review should be driven by a checklist of possible errors.</td>
</tr>
<tr>
<td>Progress reviews</td>
<td>To provide information for management about the overall progress of the project. This is both a product and a process review and is concerned with costs, plans and schedules.</td>
</tr>
<tr>
<td>Quality reviews</td>
<td>To carry out a technical analysis of product components or documentation to find faults or mismatches between the specification and the design, code or documentation. It may also be concerned with broader quality issues such as adherence to standards and other quality attributes.</td>
</tr>
</tbody>
</table>

The review process

Review functions

• Quality function - they are part of the general quality management process
• Project management function - they provide information for project managers
• Training and communication function - product knowledge is passed between development team members
Quality reviews

• Objective is the discovery of system defects and inconsistencies
• Any documents produced in the process may be reviewed
• Review teams should be relatively small and reviews should be fairly short
• Review should be recorded and records maintained

Review results

• Comments made during the review should be classified.
  – No action. No change to the software or documentation is required.
  – Refer for repair. Designer or programmer should correct an identified fault.
  – Reconsider overall design. The problem identified in the review impacts other parts of the design. Some overall judgement must be made about the most cost-effective way of solving the problem.
• Requirements and specification errors may have to be referred to the client.

Software measurement and metrics

• Software measurement is concerned with deriving a numeric value for an attribute of a software product or process
• This allows for objective comparisons between techniques and processes
• Although some companies have introduced measurement programmes, the systematic use of measurement is still uncommon
• There are few standards in this area

Software metric

• Any type of measurement which relates to a software system, process or related documentation
  – Lines of code in a program, the Fog index, number of person-days required to develop a component
• Allow the software and the software process to be quantified
• Measures of the software process or product
• May be used to predict product attributes or to control the software process

Predictor and control metrics

• A software property can be measured
• The relationship exists between what we can measure and what we want to know
• This relationship has been formalized and validated
• It may be difficult to relate what can be measured to desirable quality attributes
Internal and external attributes

- Maintainability
- Reliability
- Portability
- Usability
- Number of procedure parameters
- Cyclomatic complexity
- Program size in lines of code
- Number of error messages
- Length of user manual

The measurement process

- A software measurement process may be part of a quality control process
- Data collected during this process should be maintained as an organisational resource
- Once a measurement database has been established, comparisons across projects become possible

Product measurement process

Data collection

- A metrics programme should be based on a set of product and process data
- Data should be collected immediately (not in retrospect) and, if possible, automatically
- Three types of automatic data collection
  - Static product analysis
  - Dynamic product analysis
  - Process data collation

Automated data collection

Data accuracy

- Don’t collect unnecessary data
  - The questions to be answered should be decided in advance and the required data identified
- Tell people why the data is being collected
  - It should not be part of personnel evaluation
- Don’t rely on memory
  - Collect data when it is generated not after a project has finished
Product metrics

- A quality metric should be a predictor of product quality
- Classes of product metric
  - Dynamic metrics which are collected by measurements made of a program in execution
  - Static metrics which are collected by measurements made of the system representations
  - Dynamic metrics help assess efficiency and reliability; static metrics help assess complexity, understandability and maintainability

Dynamic and static metrics

- Dynamic metrics are closely related to software quality attributes
  - It is relatively easy to measure the response time of a system (performance attribute) or the number of failures (reliability attribute)
- Static metrics have an indirect relationship with quality attributes
  - You need to try and derive a relationship between these metrics and properties such as complexity, understandability and maintainability

Software product metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fan-in/fan-out</td>
<td>This is a measure of the number of functions that call some other function (i.e. X). Fan-out is the number of functions which are called by function X. A high value for fan-in means that X is tightly coupled to the rest of the design and changes to X will have extreme knock-on effects. A high value for fan-out suggests that the overall complexity of X may be high because of the complexity of the document. The metrics are closely related to software quality attributes.</td>
</tr>
<tr>
<td>Length of code</td>
<td>This is a measure of the size of the code of a program, the longer the code the longer the program and the more people that will have to understand and maintain it. The code may be modular, in which case, a short code (less lines of code) is preferable.</td>
</tr>
<tr>
<td>Cyclomatic complexity</td>
<td>This is a measure of the complexity of a program. The cyclomatic complexity of a program is given by the total number of conditions in the program. The cyclomatic complexity is a measure of the complexity of the program.</td>
</tr>
<tr>
<td>Number of identifiers</td>
<td>This is a measure of the number of identifiers used in a program. The longer the number of identifiers, the less likely they will be used in the program.</td>
</tr>
<tr>
<td>Depth of nesting</td>
<td>This is a measure of the depth of nesting of statements in a program. Ideally, the depth of nesting should be as shallow as possible.</td>
</tr>
<tr>
<td>Fog index</td>
<td>This is a measure of the average length of words in sentences in a program. The higher the value for the Fog index, the more difficult the document may be to understand.</td>
</tr>
</tbody>
</table>

Object-oriented metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Depth of inheritance</td>
<td>This measures the number of discrete levels in the inheritance tree where sub-classes inherit attributes and operations (methods) from super-classes. The deeper the inheritance tree, the more complex the design is.</td>
</tr>
<tr>
<td>Method fan-in/fan-out</td>
<td>This is directly related to fan-in and fan-out as described above and means essentially the same thing.</td>
</tr>
<tr>
<td>Weighted methods per class</td>
<td>This is the number of methods included in a class weighted by the complexity of each method. Therefore, a simple method may have a higher weight than a complex method. This indicates the complexity of the class. The higher the value, the more complex the class is.</td>
</tr>
<tr>
<td>Number of overriding operations</td>
<td>This is the number of operations in a class that are overridden by methods in the super-class. The higher the number, the less maintainable the class is.</td>
</tr>
</tbody>
</table>

Measurement analysis

- It is not always obvious what data means
  - Analysing collected data is very difficult
- Professional statisticians should be consulted if available
- Data analysis must take local circumstances into account

Measurement surprises

- Reducing the number of faults in a program leads to an increased number of help desk calls
  - The program is now thought of as more reliable and so has a wider more diverse market. The percentage of users who call the help desk may have decreased but the total may increase
  - A more reliable system is used in a different way from a system where users work around the faults. This leads to more help desk calls
Key points

• Software quality management is concerned with ensuring that software meets its required standards.
• Quality assurance procedures should be documented in an organisational quality manual.
• Software standards are an encapsulation of best practice.
• Reviews are the most widely used approach for assessing software quality.

Key points

• Software measurement gathers information about both the software process and the software product.
• Product quality metrics should be used to identify potentially problematical components.
• There are no standardised and universally applicable software metrics.

Chapter 27

Software Change

Software change

• Managing the processes of software system change

Objectives

• To explain different strategies for changing software systems
  - Software maintenance
  - Architectural evolution
  - Software re-engineering
• To explain the principles of software maintenance
• To describe the transformation of legacy systems from centralised to distributed architectures

Topics covered

• Program evolution dynamics
• Software maintenance
• Architectural evolution
Software change

- Software change is inevitable
  - New requirements emerge when the software is used
  - The business environment changes
  - Errors must be repaired
  - New equipment must be accommodated
  - The performance or reliability may have to be improved
- A key problem for organisations is implementing and managing change to their legacy systems

Software change strategies

- Software maintenance
  - Changes are made in response to changed requirements but the fundamental software structure is stable
- Architectural transformation
  - The architecture of the system is modified generally from a centralised architecture to a distributed architecture
- Software re-engineering
  - No new functionality is added to the system but it is restructured and reorganised to facilitate future changes
- These strategies may be applied separately or together

Program evolution dynamics

- Program evolution dynamics is the study of the processes of system change
- After major empirical study, Lehman and Belady proposed that there were a number of ‘laws’ which applied to all systems as they evolved
- There are sensible observations rather than laws. They are applicable to large systems developed by large organisations. Perhaps less applicable in other cases

Lehman’s laws

<table>
<thead>
<tr>
<th>Law</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Continuing change</td>
<td>A program that is used in a real-world environment necessarily must change or become progressively less useful in that environment.</td>
</tr>
<tr>
<td>Increasing-complexity</td>
<td>As an evolving program changes, its structure tends to become more complex. Extra resources must be devoted to preserving and simplifying the structure.</td>
</tr>
<tr>
<td>Large program evolution</td>
<td>Program evolution is a self-regulating process. The attributes of such a process are a compromise between releasing small programs and delaying release to accumulate resources.</td>
</tr>
<tr>
<td>Organisational stability</td>
<td>Over a program’s lifetime, its rate of development is approximately constant and independent of the resources devoted to system development.</td>
</tr>
<tr>
<td>Conservation familiarity</td>
<td>Over the lifetime of a system, the incremental change in each release is approximately constant.</td>
</tr>
</tbody>
</table>

Applicability of Lehman’s laws

- This has not yet been established
- They are generally applicable to large, tailored systems developed by large organisations
- It is not clear how they should be modified for:
  - Shrink-wrap software products
  - Systems that incorporate a significant number of COTS components
  - Small organisations
  - Medium sized systems

Software maintenance

- Modifying a program after it has been put into use
- Maintenance does not normally involve major changes to the system’s architecture
- Changes are implemented by modifying existing components and adding new components to the system
Maintenance is inevitable

- The system requirements are likely to change while the system is being developed because the environment is changing. Therefore a delivered system won’t meet its requirements!
- Systems are tightly coupled with their environment. When a system is installed in an environment it changes that environment and therefore changes the system requirements.
- Systems MUST be maintained therefore if they are to remain useful in an environment

Types of maintenance

- Maintenance to repair software faults
  - Changing a system to correct deficiencies in the way it meets its requirements
- Maintenance to adapt software to a different operating environment
  - Changing a system so that it operates in a different environment (computer, OS, etc.) from its initial implementation
- Maintenance to add to or modify the system’s functionality
  - Modifying the system to satisfy new requirements

Distribution of maintenance effort

- Fault repair (17%)
- Software adaptation (18%)
- Functionality addition or modification (65%)

Spiral maintenance model

- Usually greater than development costs (2* to 100* depending on the application)
- Affected by both technical and non-technical factors
- Increases as software is maintained.
  - Maintenance corrupts the software structure so makes further maintenance more difficult.
- Ageing software can have high support costs (e.g. old languages, compilers etc.)
Maintenance cost factors

- Team stability
  - Maintenance costs are reduced if the same staff are involved with them for some time
- Contractual responsibility
  - The developers of a system may have no contractual responsibility for maintenance so there is no incentive to design for future change
- Staff skills
  - Maintenance staff are often inexperienced and have limited domain knowledge
- Program age and structure
  - As programs age, their structure is degraded and they become harder to understand and change

Evolutionary software

- Rather than think of separate development and maintenance phases, evolutionary software is software that is designed so that it can continuously evolve throughout its lifetime

The maintenance process

Change requests

- Change requests are requests for system changes from users, customers or management
  - In principle, all change requests should be carefully analysed as part of the maintenance process and then implemented
  - In practice, some change requests must be implemented urgently
    - Fault repair
    - Changes to the system’s environment
    - Urgently required business changes

Change implementation

Emergency repair
Maintenance prediction

- Maintenance prediction is concerned with assessing which parts of the system may cause problems and have high maintenance costs
  - Change acceptance depends on the maintainability of the components affected by the change
  - Implementing changes degrades the system and reduces its maintainability
  - Maintenance costs depend on the number of changes and costs of change depend on maintainability

Change prediction

- Predicting the number of changes requires and understanding of the relationships between a system and its environment
- Tightly coupled systems require changes whenever the environment is changed
- Factors influencing this relationship are
  - Number and complexity of system interfaces
  - Number of inherently volatile system requirements
  - The business processes where the system is used

Complexity metrics

- Predictions of maintainability can be made by assessing the complexity of system components
- Studies have shown that most maintenance effort is spent on a relatively small number of system components
- Complexity depends on
  - Complexity of control structures
  - Complexity of data structures
  - Procedure and module size

Process metrics

- Process measurements may be used to assess maintainability
  - Number of requests for corrective maintenance
  - Average time required for impact analysis
  - Average time taken to implement a change request
  - Number of outstanding change requests
- If any or all of these is increasing, this may indicate a decline in maintainability

Architectural evolution

- There is a need to convert many legacy systems from a centralised architecture to a client-server architecture
- Change drivers
  - Hardware costs. Servers are cheaper than mainframes
  - User interface expectations. Users expect graphical user interfaces
  - Distributed access to systems. Users wish to access the system from different, geographically separated, computers
Distribution factors

<table>
<thead>
<tr>
<th>Factor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business importance</td>
<td>Returns on the investment of distributing a legacy system depend on its importance to the business and how long it will remain important. If distribution provides more efficient support for stable business processes then it is more likely to be a cost-effective evolution strategy.</td>
</tr>
<tr>
<td>System age</td>
<td>The older the system the more difficult it will be to modify its architecture because previous changes will have degraded the structure of the system.</td>
</tr>
<tr>
<td>System structure</td>
<td>The more modular the system, the easier it will be to change the architecture. If the application logic, the data management and the user interface of the system are closely intertwined, it will be difficult to separate functions for migration.</td>
</tr>
<tr>
<td>Hardware procurement</td>
<td>Application distribution may be necessary if there is company policy to replace expensive mainframe computers with cheaper servers.</td>
</tr>
<tr>
<td>policies</td>
<td></td>
</tr>
</tbody>
</table>

Legacy system structure

- Ideally, for distribution, there should be a clear separation between the user interface, the system services and the system data management.
- In practice, these are usually intermingled in older legacy systems.

Legacy system structures

- Layered distribution model
  - Presentation
  - Data validation
  - Interaction control
  - Application services
  - Database

Legacy system distribution

- Distribution options
  - The more that is distributed from the server to the client, the higher the costs of architectural evolution.
  - The simplest distribution model is UI distribution where only the user interface is implemented on the server.
  - The most complex option is where the server simply provides data management and application services are implemented on the client.
User interface distribution

- UI distribution takes advantage of the local processing power on PCs to implement a graphical user interface
- Where there is a clear separation between the UI and the application then the legacy system can be modified to distribute the UI
- Otherwise, screen management middleware can translate text interfaces to graphical interfaces

UI migration strategies

Key points

- Software change strategies include software maintenance, architectural evolution and software re-engineering
- Lehman’s Laws are invariant relationships that affect the evolution of a software system
- Maintenance types are
  - Maintenance for repair
  - Maintenance for a new operating environment
  - Maintenance to implement new requirements
- The costs of software change usually exceed the costs of software development
- Factors influencing maintenance costs include staff stability, the nature of the development contract, skill shortages and degraded system structure
- Architectural evolution is concerned with evolving centralised to distributed architectures
- A distributed user interface can be supported using screen management middleware