Software Quality and Measurement

Idaho State University | Computer Science

Isaac Griffith, Ph.D.

CS 3321
Department of Computer Science
Idaho State University

ROAR

Idaho State University
Computer Science

Topics Covered

- Software Quality
- Software Measurement

ROAR

Software Quality

CS 3321

ROAR

Idaho State University
Computer Science

Software quality management

- Concerned with ensuring that the required level of quality is achieved in a software product.

Three principal concerns:

- **Organizational Level**
  1. establishing a framework of processes and standards that will lead to high-quality software.
- **Project Level**
  2. application of specific quality processes
  3. establishing a quality plan for a project
     - Includes project quality goals
     - Defines processes and standards to be used.

ROAR

Scope of quality management

- Quality management is particularly important for large, complex systems. The quality documentation is a record of progress and supports continuity of development as the development team changes.
- For smaller systems, quality management needs less documentation and should focus on establishing a quality culture.
- Techniques have to evolve when agile development is used.

Software quality

- Quality, simplistically, means that a product should meet its specification.
- This is problematical for software systems
  - There is a 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 focus may be ‘fitness for purpose’ rather than specification conformance.

Software fitness for purpose

- Has the software been properly tested?
- Is the software sufficiently dependable to be put into use?
- Is the performance of the software acceptable for normal use?
- Is the software usable?
- Is the software well-structured and understandable?
- Have programming and documentation standards been followed in the development process?

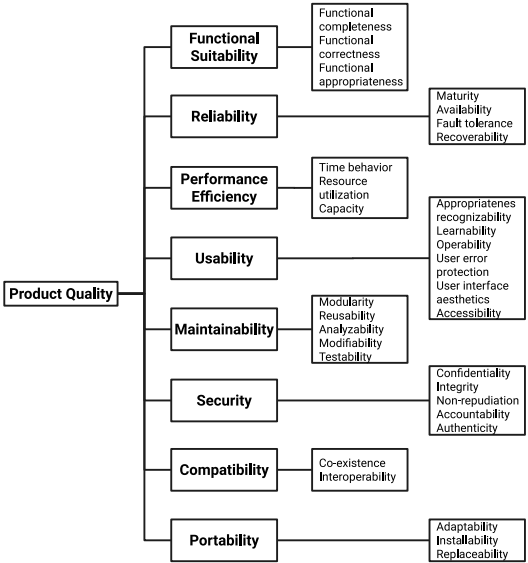
Idaho State University
Computer Science

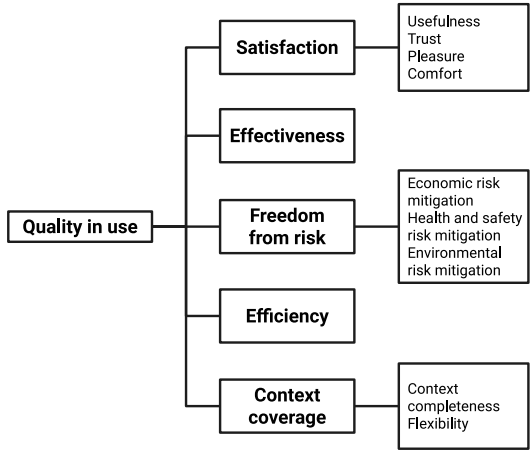
Non-functional characteristics

- The subjective quality of a software system is largely based on its non-functional characteristics.
- This reflects practical user experience – if the software's functionality is not what is expected, then users will often just work around this and find other ways to do what they want to do.
- However, if the software is unreliable or too slow, then it is practically impossible for them to achieve their goals.

Idaho State University
Computer Science

Software Quality and ISO/IEC 25010





ROAR

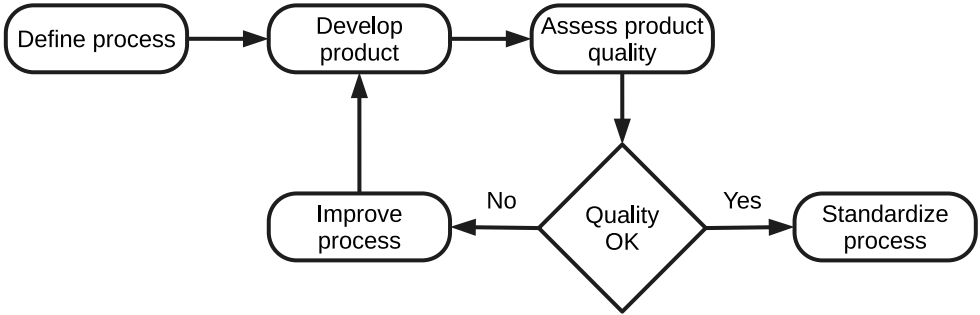
Quality conflicts

- It is not possible for any system to be optimized for all of these attributes – for example, improving robustness may lead to loss of performance.
- The quality plan should therefore define the most important quality attributes for the software that is being developed.
- The plan should also include a definition of the quality assessment process, an agreed way of assessing whether some quality, such as maintainability or robustness, is present in the product.

Process and product quality

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

Process-based quality



Quality culture

- Quality managers should aim to develop a ‘quality culture’ where everyone responsible for software development is committed to achieving a high level of product quality.
- They should encourage teams to take responsibility for the quality of their work and to develop new approaches to quality improvement.
- They should support people who are interested in the intangible aspects of quality and encourage professional behavior in all team members.

Software Measurement

CS 3321

ROAR

Software measurement

- 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 programs, most organizations still don’t make systematic use of software measurement.
- There are few established standards in this area.

Software metric

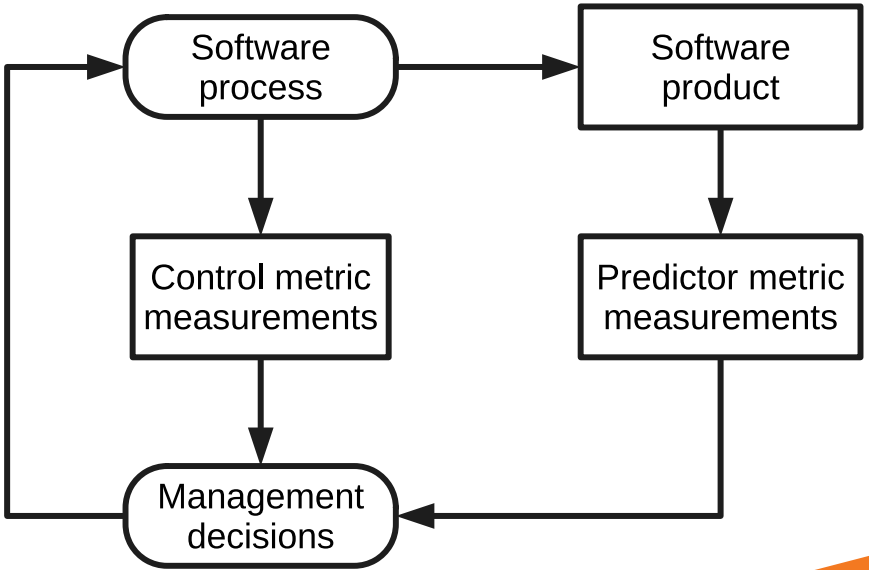
- Any type of measurement which relates to a software system, process or related documentation
  - Lines of code in a program, the Fog index, number of person-days required to develop a component.
- Allow the software and the software process to be quantified.
- May be used to predict product attributes or to control the software process.
- Product metrics can be used for general predictions or to identify anomalous components.

Types of process metric

- The time taken for a particular process to be completed
  - This can be the total time devoted to the process, calendar time, the time spent on the process by particular engineers, and so on.
- The resources required for a particular process
  - Resources might include total effort in person-days, travel costs or computer resources.
- The number of occurrences of a particular event
  - Examples of events that might be monitored include the number of defects discovered during code inspection, the number of requirements changes requested, the number of bug reports in a delivered system and the average number of lines of code modified in response to a requirements change.

Idaho State University
Computer Science

Predictor and control measurements



ROAR

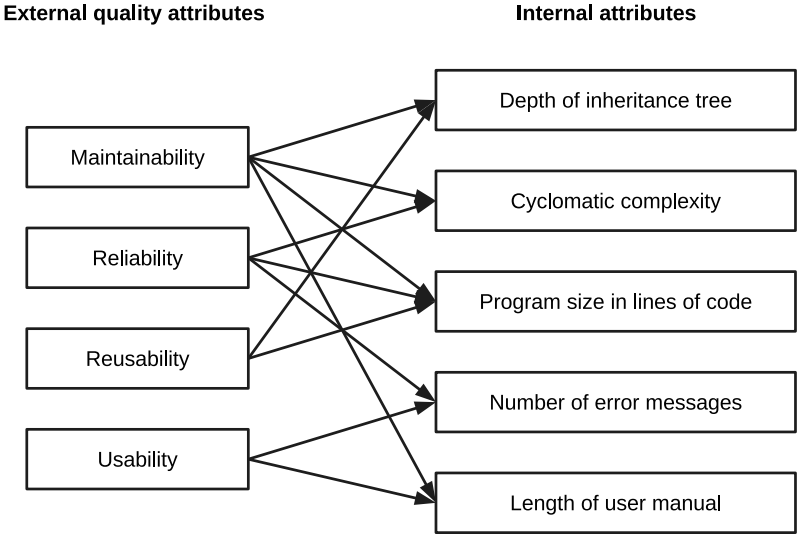
Use of measurements

- To assign a value to system quality attributes
  - By measuring the characteristics of system components, such as their cyclomatic complexity, and then aggregating these measurements, you can assess system quality attributes, such as maintainability.
- To identify the system components whose quality is sub-standard
  - Measurements can identify individual components with characteristics that deviate from the norm. For example, you can measure components to discover those with the highest complexity. These are most likely to contain bugs because the complexity makes them harder to understand.

Metrics assumptions

- A software property can be measured accurately.
- The relationship exists between what we can measure and what we want to know. We can only measure internal attributes but are often more interested in external software attributes.
- This relationship has been formalized and validated.
- It may be difficult to relate what can be measured to desirable external quality attributes.

Software attribute relationships



Problems with measurement in industry

- It is impossible to quantify the return on investment of introducing an organizational metrics program.
- There are no standards for software metrics or standardized processes for measurement and analysis.
- In many companies, software processes are not standardized and are poorly defined and controlled.
- Most work on software measurement has focused on code-based metrics and plan-driven development processes. However, more and more software is now developed by configuring ERP systems or COTS.
- Introducing measurement adds additional overhead to processes.

Empirical software engineering

- Software measurement and metrics are the basis of empirical software engineering.
- This is a research area in which experiments on software systems and the collection of data about real projects has been used to form and validate hypotheses about software engineering methods and techniques.
- Research on empirical software engineering, this has not had a significant impact on software engineering practice.
- It is difficult to relate generic research to a project that is different from the research study.

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.

Static software metrics

- **Fan-in/Fan-out**:
  - Fan-in - a measure of the number of functions or methods that call another function or method (say X).
  - Fan-out - the number of functions that 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 extensive 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 control logic needed to coordinate the called components.
- **Length of code** - This is a measure of the size of a program.
  - Generally, the larger the size of the code of a component, the more complex and error-prone that component is likely to be.
  - Length of code has been shown to one of the most reliable metrics for predicting error-proneness in components.

Static software metrics

- **Cyclomatic complexity** - This is a measure of t he control complexity of a program. This control complexity may be related to program understandability.
  - This metric is further discussed in Chapter 8.
- **Length of identifiers** - a measure of the average length of identifiers (names for variables, classes, methods, etc.) in a program
  - The longer the identifiers, the more likely they are to be meaningful and hence the more understandable the program.
- **Depth of conditional nesting** - This is a measure of the depth of nesting of if-statements in a program.
  - Deeply nested if-statements are hard to understand and potentially error-prone.
- **Fog index** - This is a measure of the average length of words and sentences in documents.
  - The higher the value of a document's Fog index, the more difficult the document is to understand.

The CK object-oriented metrics suite

- **Weighted methods per class (WMC)** - This is the number of methods in each class, weighted by the complexity of each method. Therefore, a simple method may have a complexity of 1, and a large and complex method a much higher value.
  - The larger the value for this metric, the more complex the object class. Complex objects are more likely to be difficult to understand.
  - They may not be logically cohesive, so cannot be reused effectively as superclasses in an inheritance tree.
- **Depth of inheritance tree (DIT)** - This represents the number of discrete levels in the inheritance tree where subclasses inherit attributes and operations (methods) from superclasses.
  - The deeper the inheritance tree, the more complex the design.
  - Many object classes may have to be understood to undersand the object classes at the leaves of the tree.

The CK object-oriented metrics suite

- **Number of children (NOC)** - This is a measure of the number of immediate subclasses in a class. It measures the breadth of a class hierarchy, whereas DIT measures its depth.
  - A high value for NOC may indicate greater reuse.
  - It may mean that more effort should be made in validating base classes because of the number of subclasses that depend on them.
- **Coupling between object classes (CBO)** - Classes are coupled when methods in one class use methods or instance variables in a different class. CBO is a measure of how much coupling exists.
  - A high value for CBO means that classes are highly dependent, and therefore it is more likely that changing one class will affect other classes in the program.

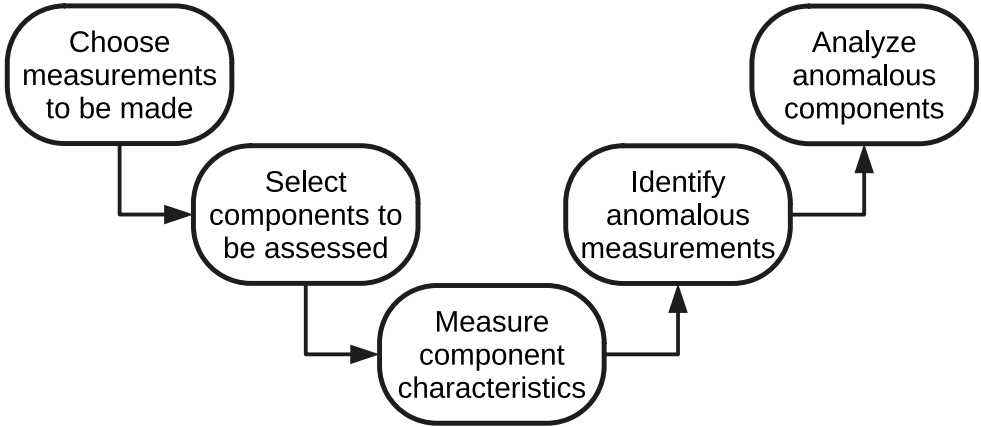
The CK object-oriented metrics suite

- **Response for a class (RFC)** - RFC is a measure of the number of methods that could potentially be executed in response to a message received by an object of that class. Again, RFC is related to complexity.
  - The higher the value for RFC, the more complex a class and hence the more likely it is that it will include errors.
- **Lack of cohesion in methods (LCOM)** - LCOM is calculated by considering pairs of methods in a class. LCOM is the difference between the number of method pairs without shared attributes and the number of method pairs with shared attributes.
  - The value of this metric has been widely debated and it exists in several variations.
  - It is not clear if it really adds any additional, useful information over and above that provided by other metrics

Software component analysis

- System component can be analyzed separately using a range of metrics.
- The values of these metrics may then compared for different components and, perhaps, with historical measurement data collected on previous projects.
- Anomalous measurements, which deviate significantly from the norm, may imply that there are problems with the quality of these components.

The process of product measurement



Measurement ambiguity

- When you collect quantitative data about software and software processes, you have to analyze that data to understand its meaning.
- It is easy to misinterpret data and to make inferences that are incorrect.
- You cannot simply look at the data on its own. You must also consider the context where the data is collected.

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.

Software context

- Processes and products that are being measured are not insulated from their environment.
- The business environment is constantly changing and it is impossible to avoid changes to work practice just because they may make comparisons of data invalid.
- Data about human activities cannot always be taken at face value. The reasons why a measured value changes are often ambiguous. These reasons must be investigated in detail before drawing conclusions from any measurements that have been made.

Key points

- Software measurement can be used to gather quantitative data about software and the software process.
- You may be able to use the values of the software metrics that are collected to make inferences about product and process quality.
- Product quality metrics are particularly useful for highlighting anomalous components that may have quality problems. These components should then be analyzed in more detail.
- Software analytics is the automated analysis of large volumes of software product and process data to discover relationships that may provide insights for project managers and developers.