605.601 Foundations of Software Engineering
Fall 2020

Module 10: Process Improvement and Quality

Dr. Tushar K. Hazra

tkhazra@gmail.com

(443)540-2230

Course Module 10: Quality

Topics for Discussion

- Concepts – Quality
- Software Quality
- Different Types of Metrics

Quality

- The American Heritage Dictionary defines quality as "a characteristic or attribute of something."
- For software, two kinds of quality may be encountered: Quality of design Encompasses requirements, specifications, and the design of the system
- Quality of conformance An issue focused primarily on implementation

- User satisfaction = compliant product + good quality + delivery within budget and schedule

Quality: A Philosophical View

- Robert Pirsig commented on the thing we call quality [Pirsig, 1974]: Quality. . . you know what it is, yet you don't know what it is. But that's self-contradictory. But some things are better than others, that is, they have more quality. But when you try to say what the quality is, apart from the things that have it, it all goes poof! There's nothing to talk about.

Course Module 10: Quality

Quality: A Pragmatic View

- The transcendental view argues (like Persig) that quality is something that you immediately recognize, but cannot explicitly define.
- The user view sees quality in terms of an end user's specific goals. If a product meets those goals, it exhibits quality.
- The manufacturer's view defines quality in terms of the original specification of the product. If the product conforms to the spec, it exhibits quality.
- The product view suggests that quality can be tied to inherent characteristics (e.g., functions and features) of a product.
- Finally, the value-based view measures quality based on how much a customer is willing to pay for a product.
- In reality, quality encompasses all of these views and more.

Software Quality

- Software quality is an effective software process applied in a manner that creates a useful product that provides measurable value for those who produce it and those who use it.

Effective Software Process

- An effective software process establishes the infrastructure that supports any effort at building a high quality software product.
- The management aspects of process create the checks and balances that help avoid project chaos—a key contributor to poor quality.
- Software engineering practices allow the developer to analyze the problem and design a solid solution—both are critical to building high quality software.
- Finally, umbrella activities such as change management and technical reviews have as much to do with quality as any other part of software engineering practice.

Useful Product

- A useful product delivers the content, functions, and features that the end user desires.
- But as important, it delivers these assets in a reliable, error-free way.
- A useful product always satisfies those requirements that have been explicitly stated by stakeholders.
- In addition, it satisfies a set of implicit requirements (e.g., ease of use) that are expected of all high quality software.

Adding Value

- High quality software provides benefits for the software organization and users
- The software organization gains added value because high quality software requires less maintenance effort, fewer bug fixes, and reduced customer support.

Adding Value (continued)

- The user community gains added value because the application provides a useful capability in a way that expedites some business process.
- The end result is. . .
  - .., greater software product revenue,
  - .., better profitability when an application supports a business process, or
  - .., improved availability of information that is crucial for the business.

Quality Dimensions I

- Performance Does the software deliver all content, functions, and features that are specified as part of the requirements model in a way that provides value to the end user?
- Features Does the software provide features that surprise and delight first-time end users?
- Reliability Does the software deliver all features and capability without failure? Is it available when it is needed? Does it deliver functionality that is error free?

Quality Dimensions II

- Conformance Does the software conform to local and external software standards that are relevant to the application? Does it conform to de facto design and coding conventions? For example, does the user interface conform to accepted design rules for menu selection or data input?
- Durability Can the software be maintained (changed) or corrected (debugged) without the inadvertent generation of unintended side effects? Will changes cause the error rate or reliability to degrade with time?
- Serviceability Can the software be maintained (changed) or corrected (debugged) in an acceptably short time period. Can support staff acquire all information they need to make changes or correct defects?

Quality Dimensions III

- Aesthetics Most of us would agree that an aesthetic entity has a certain elegance, a unique flow, and an obvious "presence" that are hard to quantify but evident nonetheless.
- Perception In some situations, you have a set of prejudices that will influence your perception of quality.

Cost of Quality

- Prevention costs include
  - Quality planning
  - Formal technical reviews
  - Test equipment
  - Training
- Internal failure costs include
  - Rework
  - Repair
  - Failure mode analysis

Course Module 10: Quality

Cost of Quality (continued)

- External failure costs are
  - Complaint resolution
  - Product return and replacement
  - Help line support
  - Warranty work

Quality and Security

- Software security relates entirely and completely to quality. You must think about security, reliability, availability, dependability—at the beginning, in the design, architecture, test, and coding phases, all through the software life cycle [process]. Even people aware of the software security problem have focused on late life-cycle stuff. The earlier you find the software problem, the better. And there are two kinds of software problems. One is bugs, which are implementation problems. The other is software flaws—architectural problems in the design. People pay too much attention to bugs and not enough on flaws. (Gary McGraw)

Achieving Software Quality

- Critical success factors:
  - Software Engineering Methods
  - Project Management Techniques
  - Quality Control
  - Quality Assurance

605.601 Foundations of Software Engineering Course Module 10: Quality

- McCall's Triangle of Quality

Image source: Pressman, R. (2010). *Software Engineering: A Practitioner's Approach*. McGraw-Hill, Inc., New York, NY, 7th edition

Measures, Metrics, and Indicators

- A measure provides a quantitative indication of the extent, amount, dimension, capacity, or size of some attribute of a product or process.
- The IEEE glossary defines a metric as "a quantitative measure of the degree to which a system, component, or process possesses a given attribute."
- An indicator is a metric or combination of metrics that provide insight into the software process, a software project, or the product itself.

Measurement Principles

- The objectives of measurement should be established before data collection begins
- Each technical metric should be defined in an unambiguous manner
- Metrics should be derived based on a theory that is valid for the domain of application (e.g., metrics for design should draw upon basic design concepts and principles and attempt to provide an indication of the presence of an attribute that is deemed desirable)
- Metrics should be tailored to best accommodate specific products and processes

Measurement Process

- Formulation The derivation of software measures and metrics appropriate for the representation of the software that is being considered
- Collection The mechanism used to accumulate data required to derive the formulated metrics
- Analysis The computation of metrics and the application of mathematical tools
- Interpretation The evaluation of metrics results in an effort to gain insight into the quality of the representation
- Feedback Recommendations derived from the interpretation of product metrics transmitted to the software team

Goal-Oriented Software Measurement

- Goal / Question / Metric Paradigm
- Goal Establish an explicit measurement goal that is specific to the process activity or product characteristic that is to be assessed
- Question Define a set of questions that must be answered in order to achieve the goal
- Metric Identify well-formulated metrics that help to answer these questions

Goal-Oriented Software Measurement (continued)

- Goal Definition Template
- Analyze the name of activity or attribute to be measured
- for the purpose of the overall objective of the analysis
- with respect to the aspect of the activity or attribute that is considered
- from the viewpoint of the people who have an interest in the measurement
- in the context of the environment in which the measurement takes place.

Metrics Attributes

- Simple and computable: It should be relatively easy to learn how to derive the metric, and its computation should not demand inordinate effort or time.
- Empirically and intuitively persuasive: The metric should satisfy the engineer's intuitive notions about the product attribute under consideration.

Metrics Attributes (Continued)

- Consistent and objective: The metric should always yield results that are unambiguous.
- Consistent in its use of units and dimensions: The mathematical computation of the metric should use measures that do not lead to bizarre combinations of unit.
- Programming language independent: Metrics should be based on the analysis model, the design model, or the structure of the program itself.

Collection and Analysis Principles

- Whenever possible, data collection and analysis should be automated
- Valid statistical techniques should be applied to establish relationship between internal product attributes and external quality characteristics
- Interpretative guidelines and recommendations should be established for each metric

Metrics for the Requirements Model

- Function-based metrics use the function point as a normalizing factor or as a measure of the "size" of the specification
- Specification metrics used as an indication of quality by measuring number of requirements by type

Function-Based Metrics

- The function point metric (FP), first proposed by Albrecht, can be used effectively as a means for measuring the functionality delivered by a system.
- Function points are derived using an empirical relationship based on countable (direct) measures of software's information domain and assessments of software complexity.

Function-Based Metrics (continued)

- Information domain values are defined in the following manner:
  - .., number of external inputs (EIs)
  - .., number of external outputs (EOs)
  - .., number of external inquiries (EQs)
  - .., number of internal logical files (ILFs)
  - .., number of external interface files (EIFs)

Architectural Design Metrics

- Architectural design metrics
  - .., Structural complexity = g(fan-out)
  - .., Data complexity = f (input & output variables, fan-out)
  - .., System complexity = h(structural & data complexity)
- HK metric is architectural complexity as a function of fan-in and fan-out
- Morphology metrics is a function of the number of modules and the number of interfaces between modules

Metrics for OO Design I

- Size Four views: population, volume, length, and functionality
- Complexity How classes of an object-oriented design are interrelated to one another
- Coupling The physical connections between elements of the object-oriented design
- Sufficiency "the degree to which an abstraction possesses the features required of it, or the degree to which a design component possesses features in its abstraction, from the point of view of the current application"

Metrics for OO Design II

- Completeness An indirect implication about the degree to which the abstraction or design component can be reused
- Cohesion The degree to which all operations working together to achieve a single, well-defined purpose
- Primitiveness Applied to both operations and classes, the degree to which an operation is atomic
- Similarity The degree to which two or more classes are similar in terms of their structure, function, behavior, or purpose
- Volatility Measures the likelihood that a change will occur

Distinguishing Characteristics

- Localization The way in which information is concentrated in a program
- Encapsulation The packaging of data and processing Information hiding The way in which information about
- operational details is hidden by a secure interface
- Inheritance The manner in which the responsibilities of one class are propagated to another
- Abstraction The mechanism that allows a design to focus on essential details

Course Module 10: Quality

Metrics for Testing

- Testing effort can also be estimated using metrics derived from Halstead measures
- Binder suggests a broad array of design metrics that have a direct influence on the "testability" of an object-oriented system
  - .., Lack of cohesion in methods
  - .., Percent public and protected
  - .., Public access to data members
  - .., Number of root classes
  - .., Fan-in
  - .., Number of children and depth of the inheritance tree