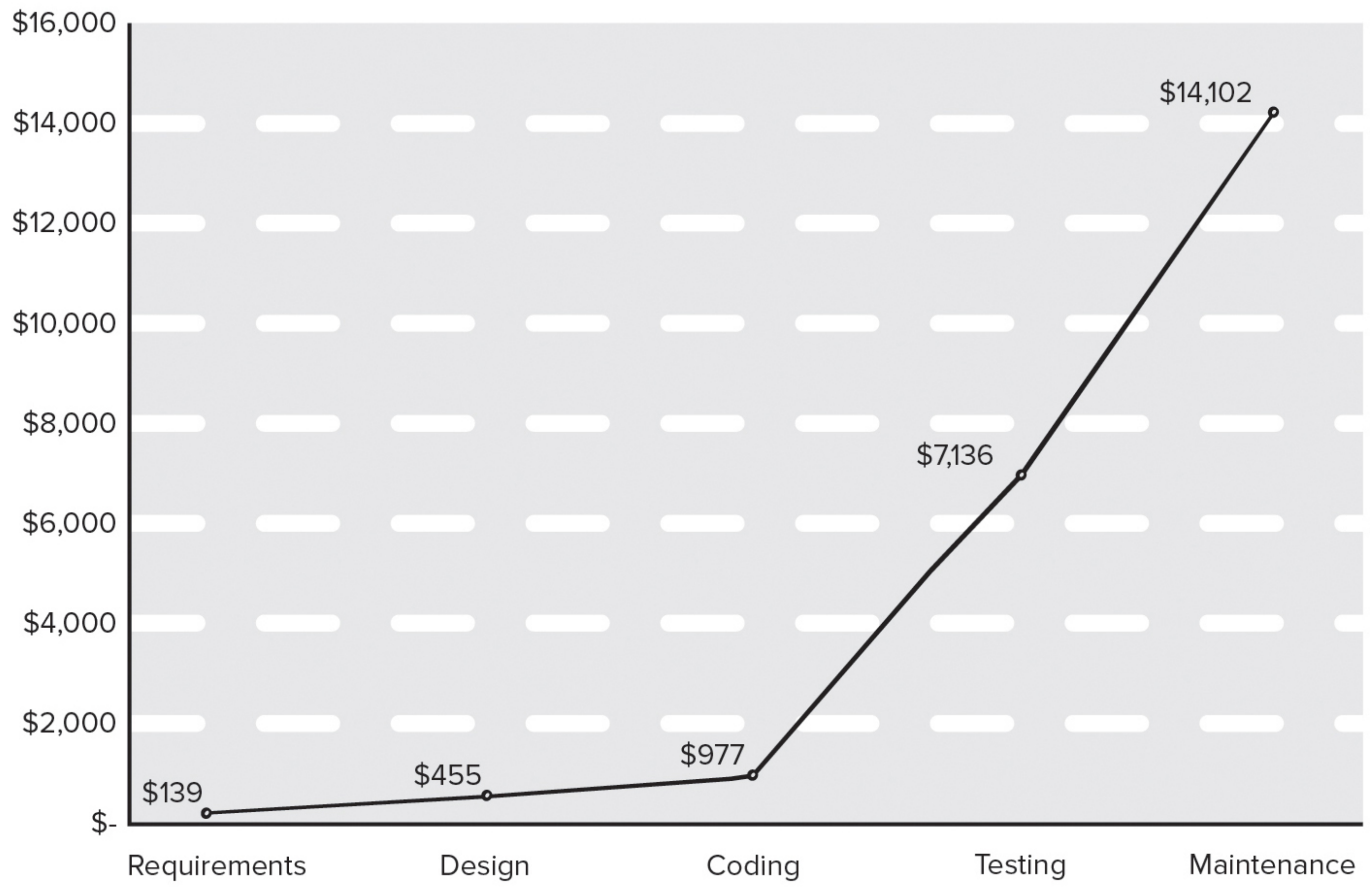
COMP 354: Introduction to Software Engineering

Software Quality Concepts

Based on Chapter 15 of the textbook

Relative Costs to Find and Repair a Defect



Source: Boehm, Barry and Basili, Victor R., "Software Defect Reduction Top 10 List," IEEE Computer, vol. 34, no. 1, January 2001.

What is Quality?

- The *American Heritage Dictionary* defines quality as “a characteristic or attribute of something.”
- For software, three kinds of quality may be encountered:
  - Quality of design encompasses requirements, specifications, and the design of the system.
  - Quality of conformance is an issue focused primarily on implementation.
  - User satisfaction = compliant product + good quality + delivery within budget and schedule.

Quality – Philosophical View

Robert Persig commented on the thing we call quality:

- 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.
- But if you can't say what Quality is, how do you know what it is, or how do you know that it even exists? If no one knows what it is, then for all practical purposes it doesn't exist at all.
- But for all practical purposes it really does exist. What else are the grades based on? Why else would people pay fortunes for some things and throw others in the trash pile.
- What the hell is Quality? What is it?

Quality – Pragmatic Views

- The transcendental view argues that quality is something that you immediately recognize but cannot explicitly define.
- The user view sees product quality in terms of meeting the end-user's specific goals.
- The manufacturer's view defines quality in terms of making sure a product its original specification.
- The product view suggests that quality can be tied to inherent characteristics (for example: functions and features) of a product.
- The value-based view measures quality based on how much a customer is willing to pay for a product.
- Quality encompasses all of these views and more.

Software Quality

- Software quality can be defined as:
  - 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.
- Advantages of providing useful products:
  - Greater software product revenue.
  - Better profitability when an application supports a business process.
  - Improved availability of information that is crucial for the business.

Software Quality – Effective Process

- An effective software process establishes infrastructure that supports 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 critical to building high quality software.
- Umbrella activities such as change management and technical reviews have as much to do with quality as any other part of software engineering practice.

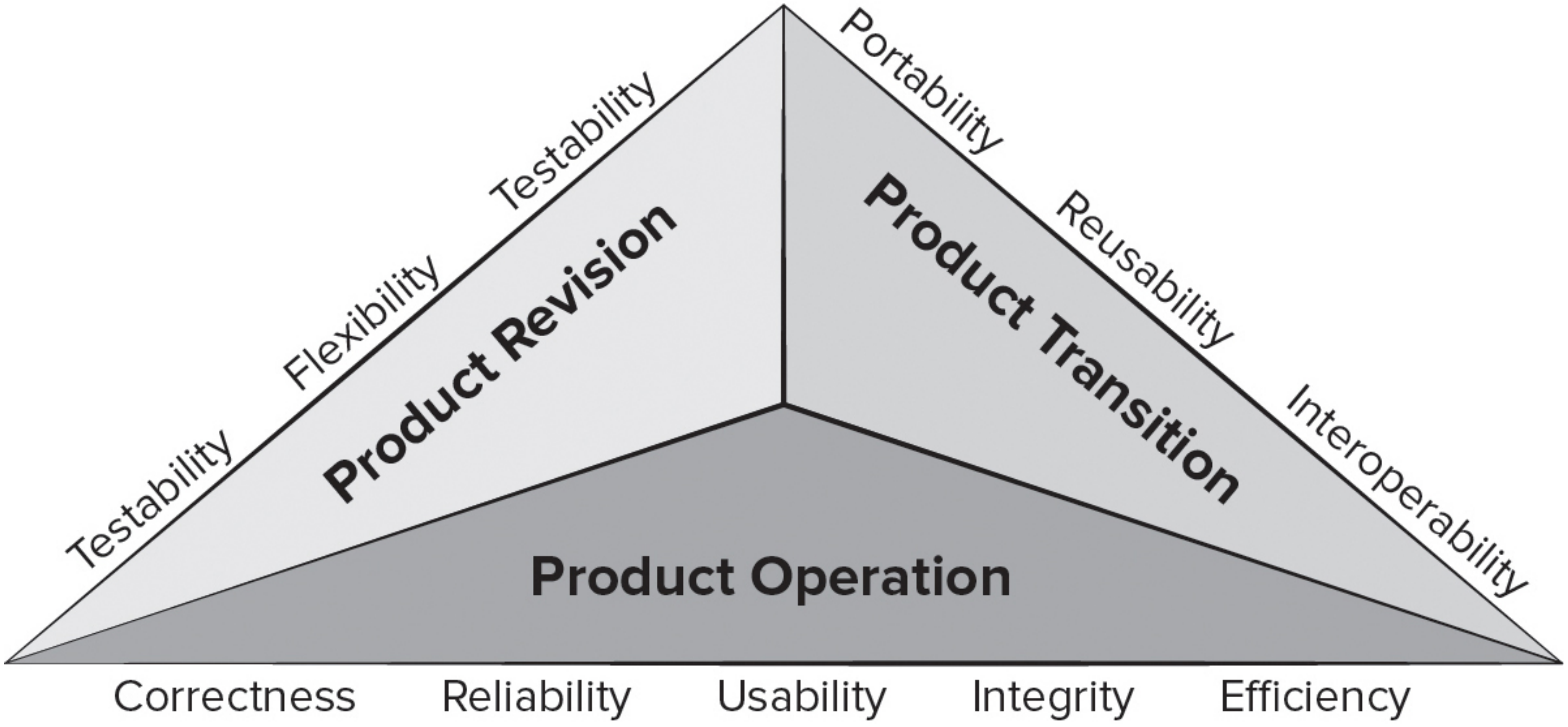
Software Quality – Useful Product

- 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.
- A useful product satisfies a set of implicit requirement that are expected of all high-quality software.

Software Quality – Adding Value

- By adding value for both the producer and user of a software product, high quality software provides benefits for the software organization and the end-user community.
- The software organization gains added value because high quality software requires less maintenance effort, fewer bug fixes, and reduced customer support.
- The user community gains added value because the application provides a useful capability in a way that expedites some business process.

McCall's Quality Factors



Quality in Use – ISO25010:2017

- Effectiveness. Accuracy and completeness with which users achieve goals.
- Efficiency. Resources expended to achieve user goals completely with desired accuracy.
- Satisfaction. Usefulness, trust, pleasure, comfort
- Freedom from risk. Mitigation of economic, health, safety, and environmental risks.
- Context coverage. Completeness, flexibility.

Product Quality – ISO25010:2017

- Functional suitability. Complete, correct, appropriate.
- Performance efficiency. Timing, resource use, capacity.
- Compatibility. Coexistence, interoperability.
- Usability. Appropriateness, learnability, operability, error protection, aesthetics, accessibility.
- Reliability. Maturity, availability, fault tolerance.
- Security. Confidentiality, integrity, authenticity.
- Maintainability. Reusability, modifiability, testability.
- Portability. Adaptability, installability, replaceability.

Qualitative Quality Assessment

- These product quality dimensions and factors presented focus on the complete software product and can be used as a generic indication of the quality of an application.
- You and your team might decide to create a user questionnaire and a set of structured tasks for users to perform for each quality factor you want to assess.
- You might observe the users while they perform these tasks and have them complete the questionnaire when they finish.
- For some quality factors it may be important to test the software in the wild (or in the production environment).

Quantitative Quality Assessment

- The software engineering community strives to develop precise measures for software quality.
- Internal code attributes can sometimes be described quantitatively using software metrics.
- Any time software metric values computed for a code fragment fall outside the range of acceptable values, it may signal the existence of a quality problem.
- Metrics represent indirect measures; we never really measure quality but rather some manifestation of quality.
- The complicating factor is the accuracy of the relationship between the variable that is measured and the quality of software.

Software Quality Dilemma

- If you produce a software system that has terrible quality, you lose because no one will want to buy it.
- If you spend infinite time, extremely large effort, and huge sums of money to build a perfect piece of software, then it's going to take so long to complete and will be so expensive to produce that you'll be out of business.
- You will either missed the market window, or you exhausted all your resources.
- People in industry try to find that magical middle ground where the product is good enough not to be rejected right away, but also not the object of so much perfectionism that it would take too long or cost too much to complete.

Good Enough Software

Arguments against “good enough” (buggy software):

- It is true that “good enough” may work in some application domains and for a few major software companies.
- If you work for a small company and you deliver a “good enough” (buggy) product, you risk permanent damage to your company’s reputation and may lose customers.
- If you work in certain application domains (for example: real time embedded software - application software that is integrated with hardware) delivering “good enough” may be considered negligent and open your company to expensive litigation.

Cost of Quality

- Prevention costs - quality planning, formal technical reviews, test equipment, training.
- Appraisal costs - conducting technical reviews, data collection and metrics evaluation, testing and debugging.
- Internal failure costs – rework, repair, failure mode analysis.
- External failure costs - complaint resolution, product return and replacement, help line support, warranty work

Negligence and Liability

- A governmental or corporate entity hires a major software developer or consulting company to analyze requirements and then design and construct a software-based "system".
- The system might support a major corporate function (for example: pension management) or some governmental function (for example: healthcare administration or homeland security).
- Work begins with the best of intentions on both sides, but by the time the system is delivered, things have gone bad.
- The system is late, fails to deliver desired features and functions, error-prone, and does get customer acceptance.
- Litigation ensues.

Quality, Risk, and Security

- Low quality software increases risks for both developers and end-users.
- When systems are delivered late, fail to deliver functionality, and does not meet customer expectations litigation ensues.
- Low quality software is easier to hack and can increase the security risks for the application once deployed.
- A secure system cannot be built without focusing on quality (security, reliability, dependability) during design.
- Low quality software is liable to contain architectural flaws as well as implementation problems (bugs).

Impact of Management Decisions

- Estimation decisions – irrational delivery date estimates cause teams to take short-cuts that can lead to reduced product quality.
- Scheduling decisions – failing to pay attention to task dependencies when creating the project schedule.
- Risk-oriented decisions – reacting to each crisis as it arises rather than building in mechanisms to monitor risks may result in products having reduced quality.

Achieving Software Quality

- Software quality is the result of good project management and solid engineering practice.
- To build high quality software you must understand the problem to be solved and be capable of creating a quality design the conforms to the problem requirements.
- Project management – project plan includes explicit techniques for quality and change management.
  - Use estimation to verify that delivery dates are achievable.
  - Schedule is understood and team avoids taking shortcuts.
  - Risk planning is conducted so problems do not breed chaos, software quality will be affected in a positive way.

Achieving Software Quality

- Project plan should include explicit techniques for quality and change management.
- Quality control - series of inspections, reviews, and tests used to ensure conformance of a work product to its specifications.
- Quality assurance - consists of the auditing and reporting procedures used to provide management with data needed to make proactive decisions.
- Defect prediction is an important part of identifying software components that may have quality concerns.
- Machine learning and statistical models may help identify relationships between metrics and defection components.