BINUS UNIVERSITY
ONLINE LEARNING

People
Innovation
Excellence

Software Engineering

Topic 5
Software Quality Assurance

BINUS UNIVERSITY ONLINE LEARNING

People Innovation Excellence

Acknowledgement

These slides have been adapted from Pressman, R.S. (2015). *Software Engineering : A Practioner's Approach. 8th ed*. McGraw-Hill Companies.Inc, Americas, New York. ISBN : 978 1 259 253157. Chapter 19 and 20

BINUS UNIVERSITY ONLINE LEARNING

Learning Objectives

LO 3 : Demonstrate the quality assurances and the potential showcase business project

People Innovation Excellence

Contents

- Software Quality
- The Software Quality Dilemma
- Achieving Software Quality
- Review Techniques
- Defect Amplification
- Review Metrics
- Informal Reviews
- Formal Technical Reviews
- Comment on Quality
- Statistical SQA
- Software Reliability

SOFTWARE QUALITY

Software Quality

In 2005, *ComputerWorld* [Hil05] lamented that

“bad software plagues nearly every organization that uses computers, causing lost work hours during computer downtime, lost or corrupted data, missed sales opportunities, high IT support and maintenance costs, and low customer satisfaction.

A year later, *InfoWorld* [Fos06] wrote about the

“the sorry state of software quality” reporting that the quality problem had not gotten any better.

Today, software quality remains an issue, but who is to blame?

Customers blame developers, arguing that sloppy practices lead to low-quality software.

Developers blame customers (and other stakeholders), arguing that irrational delivery dates and a continuing stream of changes force them to deliver software before it has been fully validated.

BINUS UNIVERSITY ONLINE LEARNING

People Innovation Excellence

Software Quality

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 is an issue focused primarily on implementation.
  - User satisfaction = compliant product + good quality + delivery within budget and schedule

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.

BINUS UNIVERSITY ONLINE LEARNING

People Innovation Excellence

Software Quality

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.*
- This definition has been adapted from [Bes04] and replaces a more manufacturing-oriented view presented in earlier editions of this book.

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 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.

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.
- In addition, it satisfies a set of implicit requirements (e.g., ease of use) that are expected of all high quality software.

BINUS UNIVERSITY ONLINE LEARNING

People Innovation Excellence

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.
- The end result is:
  - (1) greater software product revenue,
  - (2) better profitability when an application supports a business process, and/or
  - (3) improved availability of information that is crucial for the business.

Software Quality

Quality Dimensions

- David Garvin [Gar87]:
  - **Performance Quality**
  - **Feature quality**
  - **Reliability**
  - **Conformance**
  - **Durability**
  - **Serviceability**
  - **Aesthetics**
  - **Perception**

SOFTWARE QUALITY DILEMMA

BINUS UNIVERSITY ONLINE LEARNING

People Innovation Excellence

The Software Quality Dilemma

- If you produce a software system that has terrible quality, you lose because no one will want to buy it.
- If on the other hand you spend infinite time, extremely large effort, and huge sums of money to build the absolutely perfect piece of software, then it's going to take so long to complete and it will be so expensive to produce that you'll be out of business anyway.
- Either you missed the market window, or you simply exhausted all your resources.
- So people in industry try to get to that magical middle ground where the product is good enough not to be rejected right away, such as during evaluation, but also not the object of so much perfectionism and so much work that it would take too long or cost too much to complete. [Ven03]

BINUS UNIVERSITY ONLINE LEARNING

People Innovation Excellence

The Software Quality Dilemma

“Good Enough” Software

- Good enough software delivers high quality functions and features that end-users desire, but at the same time it delivers other more obscure or specialized functions and features that contain known bugs.
- Arguments *against* “good enough.”
  - It is true that “good enough” may work in some application domains and for a few major software companies. After all, if a company has a large marketing budget and can convince enough people to buy version 1.0, it has succeeded in locking them in.
  - If you work for a small company be wary of this philosophy. If you deliver a “good enough” (buggy) product, you risk permanent damage to your company’s reputation.
  - You may never get a chance to deliver version 2.0 because bad buzz may cause your sales to plummet and your company to fold.
  - If you work in certain application domains (e.g., real time embedded software, application software that is integrated with hardware can be negligent and open your company to expensive litigation.

The Software Quality Dilemma

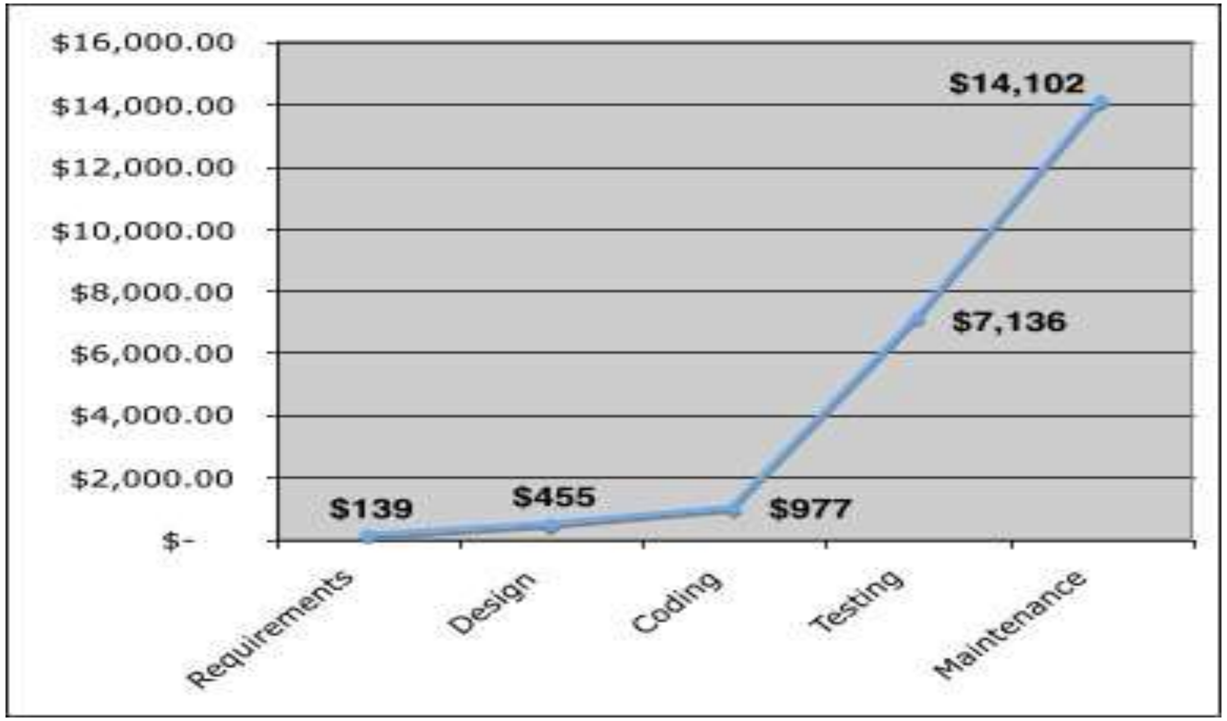
Cost of Quality

- *Prevention costs* include
  - quality planning
  - formal technical reviews
  - test equipment
  - Training
- *Internal failure costs* include
  - rework
  - repair
  - failure mode analysis
- *External failure costs* are
  - complaint resolution
  - product return and replacement
  - help line support
  - warranty work

The Software Quality Dilemma

Cost

- The relative costs to find and repair an error or defect increase dramatically as we go from prevention to detection to internal failure to external failure costs.



The Software Quality Dilemma

Quality and Risk

- *“People bet their jobs, their comforts, their safety, their entertainment, their decisions, and their very lives on computer software. It better be right.”* SEPA, Chapter 1
- Example:
  - *Throughout the month of November, 2000 at a hospital in Panama, 28 patients received massive overdoses of gamma rays during treatment for a variety of cancers. In the months that followed, five of these patients died from radiation poisoning and 15 others developed serious complications. What caused this tragedy? A software package, developed by a U.S. company, was modified by hospital technicians to compute modified doses of radiation for each patient.*

BINUS UNIVERSITY ONLINE LEARNING

The Software Quality Dilemma

Negligence and Liability

- The story is all too common. 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” to support some major activity.
  - The system might support a major corporate function (e.g., pension management) or some governmental function (e.g., 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, is error-prone, and does not meet with customer approval.
- Litigation ensues.

People Innovation Excellence

BINUS UNIVERSITY ONLINE LEARNING

People Innovation Excellence

The Software Quality Dilemma

Quality and Security

- Gary McGraw comments [Wil05]:
- “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.”

ACHIEVING SOFTWARE QUALITY

Achieving Software Quality

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

REVIEW TECHNIQUE

BINUS UNIVERSITY ONLINE LEARNING

People Innovation Excellence

Review Techniques

What Are Reviews?

- a meeting conducted by technical people for technical people
- a technical assessment of a work product created during the software engineering process
- a software quality assurance mechanism
- a training ground
- **Review are not:**
  - A project summary or progress assessment
  - A meeting intended solely to impart information
  - A mechanism for political or personal reprisal!

BINUS UNIVERSITY ONLINE LEARNING

People Innovation Excellence

Review Techniques

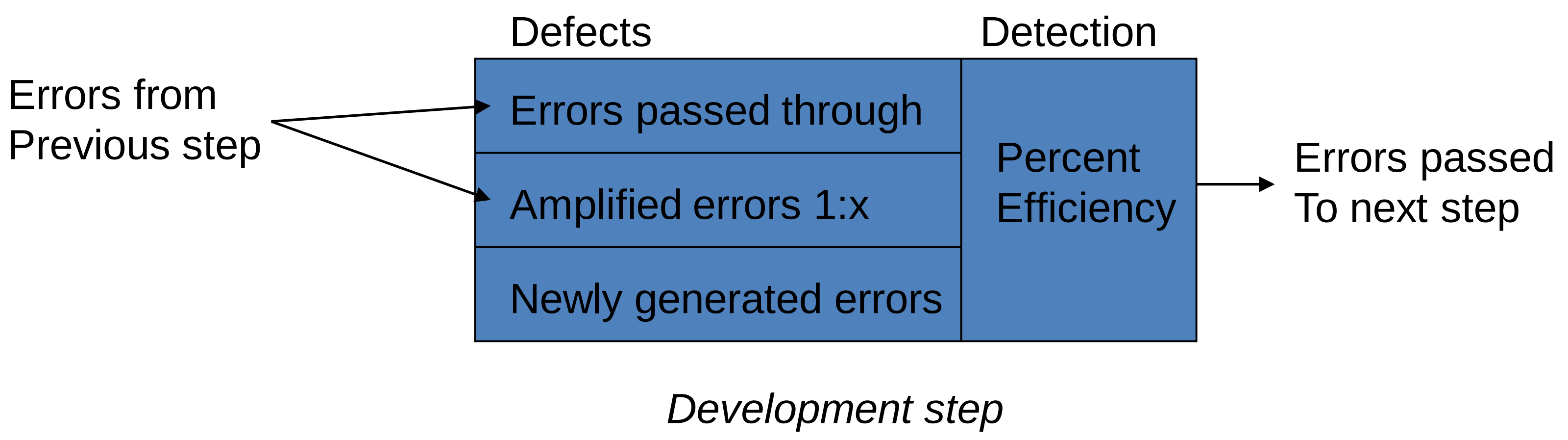
What Do We Look For?

- Errors and defects
  - *Error*—a quality problem found *before* the software is released to end users
  - *Defect*—a quality problem found only *after* the software has been released to end-users
- We make this distinction because errors and defects have very different economic, business, psychological, and human impact
- However, the temporal distinction made between errors and defects in this book is *not* mainstream thinking

People
Innovation
Excellence

Defect Amplification

- A *defect amplification model* [IBM81] can be used to illustrate the generation and detection of errors during the design and code generation actions of a software process.



BINUS UNIVERSITY ONLINE LEARNING

People Innovation Excellence

Review Metrics

- The total review effort and the total number of errors discovered are defined as:
    - $E_{review} = E_p + E_a + E_r$
    - $Err_{tot} = Err_{minor} + Err_{major}$
- *Defect density* represents the errors found per unit of work product reviewed.
    - Defect density = Err_{tot} / WPS
- where …

BINUS UNIVERSITY ONLINE LEARNING

People Innovation Excellence

Review Metrics

Metrics

- *Preparation effort, E_p*—the effort (in person-hours) required to review a work product prior to the actual review meeting
- *Assessment effort, E_a*— the effort (in person-hours) that is expending during the actual review
- *Rework effort, E_r*— the effort (in person-hours) that is dedicated to the correction of those errors uncovered during the review
- *Work product size, WPS*—a measure of the size of the work product that has been reviewed (e.g., the number of UML models, or the number of document pages, or the number of lines of code)
- *Minor errors found, Err_{minor}*—the number of errors found that can be categorized as minor (requiring less than some pre-specified effort to correct)
- *Major errors found, Err_{major}*— the number of errors found that can be categorized as major (requiring more than some pre-specified effort to correct)

BINUS UNIVERSITY ONLINE LEARNING

People Innovation Excellence

Review Metrics

An Example—I

- If past history indicates that
  - the average defect density for a requirements model is 0.6 errors per page, and a new requirement model is 32 pages long,
  - a rough estimate suggests that your software team will find about 19 or 20 errors during the review of the document.
  - If you find only 6 errors, you've done an extremely good job in developing the requirements model *or* your review approach was not thorough enough.

BINUS UNIVERSITY ONLINE LEARNING

People Innovation Excellence

Review Metrics

An Example—II

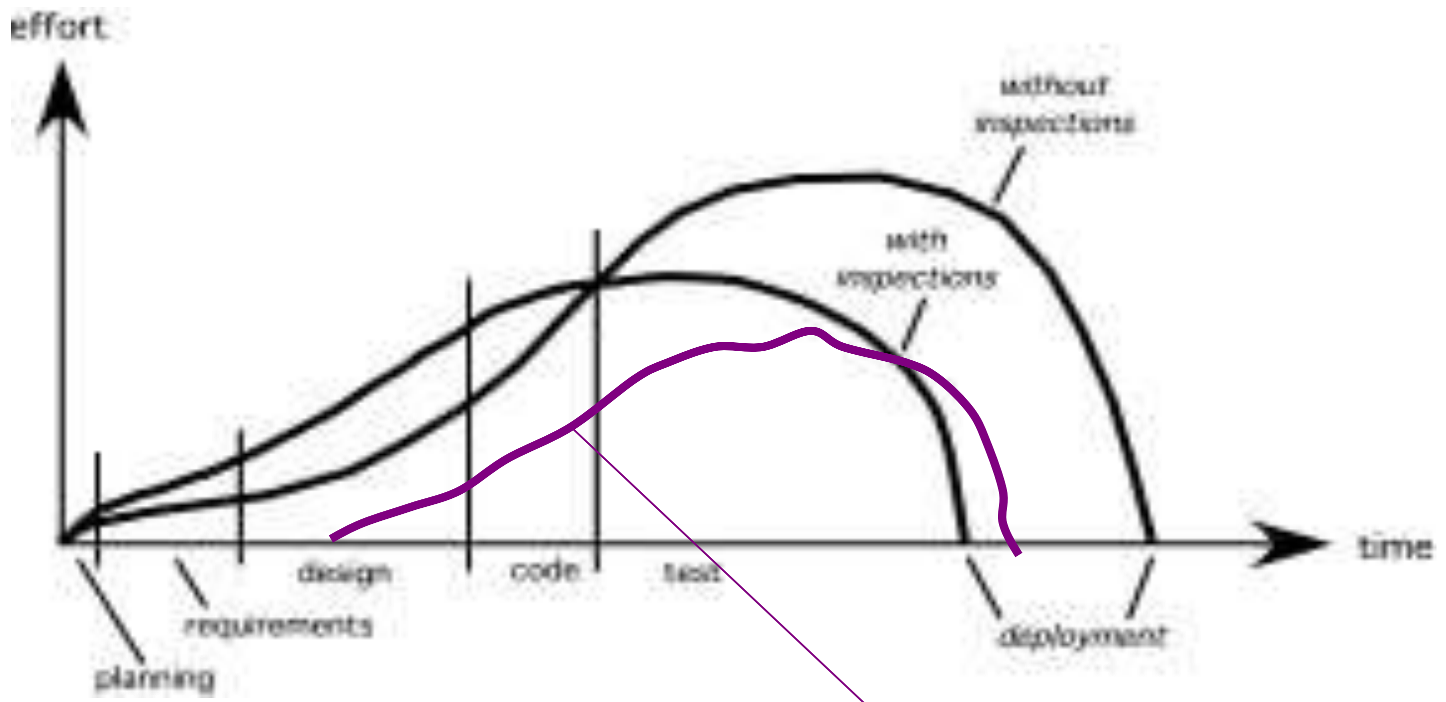
- The effort required to correct a minor model error (immediately after the review) was found to require 4 person-hours.
- The effort required for a major requirement error was found to be 18 person-hours.
- Examining the review data collected, you find that minor errors occur about 6 times more frequently than major errors. Therefore, you can estimate that the average effort to find and correct a requirements error during review is about 6 person-hours.
- Requirements related errors uncovered during testing require an average of 45 person-hours to find and correct. Using the averages noted, we get:
- Effort saved per error = $E_{testing} – E_{reviews}$
- 45 – 6 = 30 person-hours/error
- Since 22 errors were found during the review of the requirements model, a saving of about 660 person-hours of testing effort would be achieved. And that’s just for requirements-related errors.

People
Innovation
Excellence

Review Metrics

Overall

- Effort expended with and without reviews



with reviews

Informal Reviews

- Informal reviews include:
  - a simple desk check of a software engineering work product with a colleague
  - a casual meeting (involving more than 2 people) for the purpose of reviewing a work product, or
  - the review-oriented aspects of pair programming
- *pair programming* encourages continuous review as a work product (design or code) is created.
  - The benefit is immediate discovery of errors and better work product quality as a consequence.

Formal Technical Reviews

- The objectives of an FTR are:
  - to uncover errors in function, logic, or implementation for any representation of the software
  - to verify that the software under review meets its requirements
  - to ensure that the software has been represented according to predefined standards
  - to achieve software that is developed in a uniform manner
  - to make projects more manageable
- The FTR is actually a class of reviews that includes *walkthroughs* and *inspections.*

BINUS UNIVERSITY ONLINE LEARNING

People
Innovation
Excellence

Formal Technical Reviews

Conducting the Review

- *Review the product, not the producer.*
- *Set an agenda and maintain it.*
- *Limit debate and rebuttal.*
- *Enunciate problem areas, but don't attempt to solve every problem noted.*
- *Take written notes.*
- *Limit the number of participants and insist upon advance preparation.*
- *Develop a checklist for each product that is likely to be reviewed.*
- *Allocate resources and schedule time for FTRs.*
- *Conduct meaningful training for all reviewers.*
- *Review your early reviews.*

BINUS UNIVERSITY ONLINE LEARNING

People
Innovation
Excellence

References

- Pressman, R.S. (2015). ***Software Engineering : A Practioner's Approach. 8th ed***. McGraw-Hill Companies.Inc, Americas, New York. ISBN : 978 1 259 253157.
- **Introduction to software quality assurance, http://www.youtube.com/watch?v=5_cTi5xBlYg**
- Lean Six Sigma and IEEE standards for better software engineering**,** http://www.youtube.com/watch?v=oCkPD5YvWqw

BINUS UNIVERSITY ONLINE LEARNING

Q & A

People
Innovation
Excellence

BINUS
UNIVERSITY
ONLINE
LEARNING

People
Innovation
Excellence

Thank You