Software Engineering

Vinaya Sathyanarayana


Quality Concepts

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 – 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:

1. Greater software product revenue.
2. Better profitability when an application supports a business process.
3. 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.

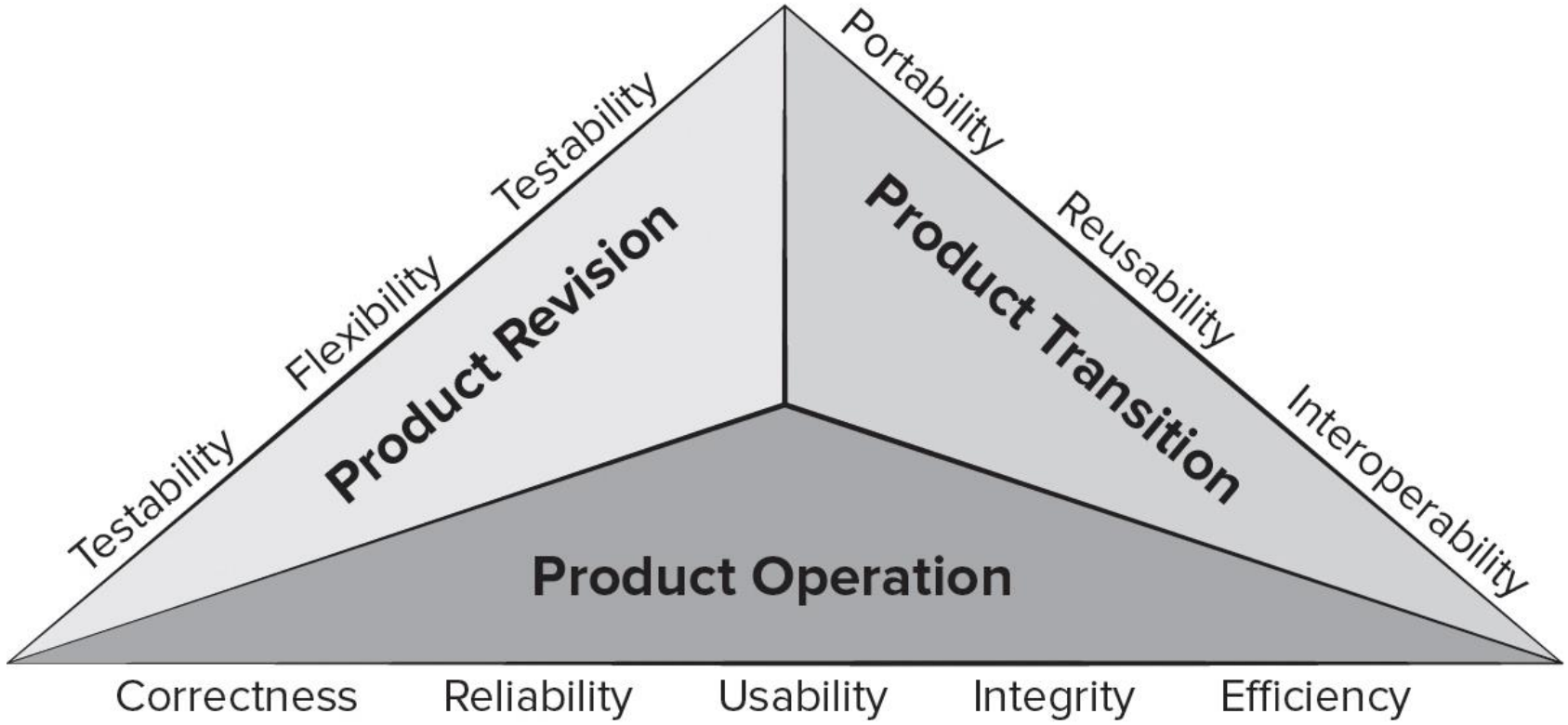
Software Quality – Useful Product

- 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



- Access the text alternative for slide images

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.

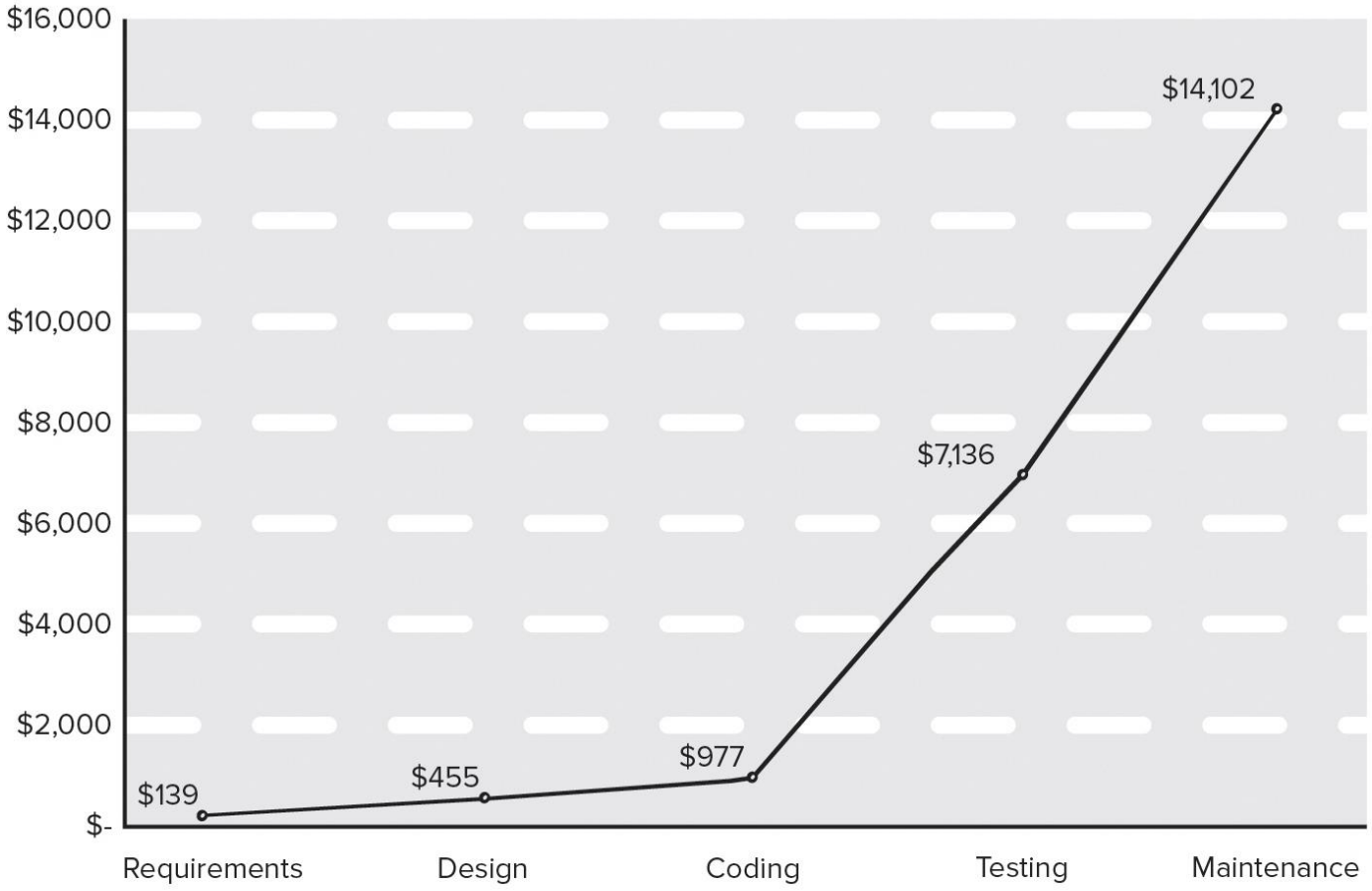
Good Enough Software

- 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

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.

- Access the text alternative for slide images.

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 [1]

- 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.

1. Use estimation to verify that delivery dates are achievable.
2. Schedule is understood and team avoids taking shortcuts.
3. Risk planning is conducted so problems do not breed chaos, software quality will be affected in a positive way.

Achieving Software Quality 2

- 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.

Reviews

Reviews

- What are they?
- A meeting conducted by technical people.
- A technical assessment of a work product created during the software engineering process.
- A software quality assurance mechanism.
- A training ground.

- What they are not!
- A project summary or progress assessment.
- A meeting intended solely to impart information.
- A mechanism for political or personal reprisal!

Rigorous Reviews

- Space Organizations (ISRO, NASA) have the most rigorous reviews

Cost Impact of Software 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.
- Design activities introduce 50 to 65% of all software defects.
- Review activities have been shown to be 75% effective in uncovering design flaws.
- The sooner you find a defect the cheaper it is to fix it.

Defect Amplification and Removal

- **Defect amplification** is a term used to describe how an defect introduced early in the software engineering work flow (for example: during requirement modeling) and undetected, can and often will be amplified into multiple errors during design and more errors in construction.
- **Defect propagation** is a term used to describe the impact an undiscovered defect has on future development activities or product behavior.
- **Technical debt** is the term used to describe the costs incurred by failing to find and fix defects early or failing to update documentation following software changes.

Review Metrics [1]

- ***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 (for example: 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

Review Metrics 2

- **Total errors found,** Errtot. Represents the sum of the errors found:

  $Err_{tot} = Err_{minor} + Err_{major}$

- **Error density.** Represents the errors found per unit of work product reviewed:

  Error density = $Err_{tot} \div$ WPS

Metrics Example [1]

- The average defect density for a requirements model is 0.68 errors per page, and a new requirement model is 40 pages long.
- A rough estimate suggests that your software team will find about 27 errors during the review of the document.
- If you find only 9 errors, you've done an extremely good job in developing the requirements model *or* your review approach was not thorough enough.

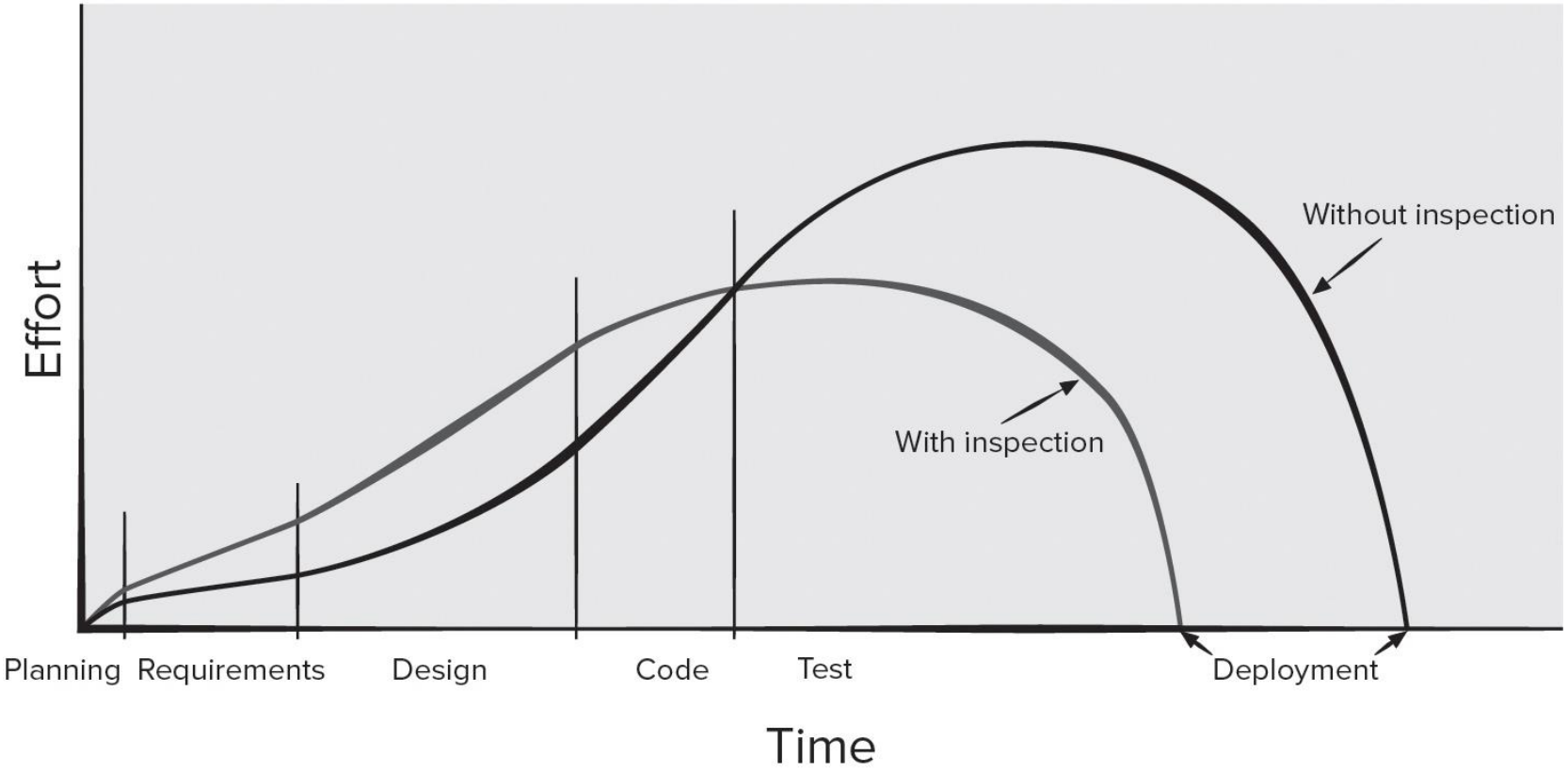
Metrics Example 2

- 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.

Metrics Example 3

- 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 $= E_{testing} - E_{reviews}$
  - $= 45 - 6 = 39$ person-hours/error
- Since 22 errors were found during the review of the requirements model, a saving of about 858 person-hours of testing effort would be achieved. And that's just for requirements-related errors.

Effort Expended With and Without Reviews



Source: Fagan, Michael E., "Advances in Software Inspections," IEEE Transactions on Software Engineering, vol. SE-12, no. 7, July 1986, 744–751.

- Access the text alternative for slide images.

Reference Model for Technical Reviews

- Access the text alternative for slide images.

Informal Reviews

- The benefit is immediate discovery of errors and better work product quality.

- 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 which encourages continuous review as work is created.

Formal Technical Reviews

- The objectives of an FTR (*walkthrough* or *inspection*) are:

1. To uncover errors in function, logic, or implementation for any representation of the software.
2. To verify that the software under review meets its requirements.
3. To ensure that the software has been represented according to predefined standards.
4. To achieve software that is developed in a uniform manner.
5. To make projects more manageable.

Review Meeting

- Between three and five people (typically) should be involved in the review.
- Advance preparation should occur but should require no more than two hours of work for each person.
- The duration of the review meeting should be less than two hours.
- Focus is on a work product (for example: a portion of a requirements model, a detailed component design, source code for a component).

Review Players

- *Producer*—the individual who has developed the work product.
- *Review leader*—evaluates the product for readiness, generates copies of product materials, and distributes them to two or three *reviewers* for advance preparation and facilitates the meeting discussion.
- *Reviewer*(*s*)—expected to spend between one and two hours reviewing the product, making notes, and otherwise becoming familiar with the work.
- *Recorder*—reviewer who records (in writing) all important issues raised during the review.

Review Outcome

- At the end of the review, all attendees of the FTR must decide whether to:

1. Accept the product without further modification.
2. Reject the product due to severe errors (once corrected, another review must be performed).
3. Accept the product provisionally (minor errors have been encountered and must be corrected, but no additional review will be required).

Review Reporting and Record Keeping

- During the FTR, the recorder records all issues raised and summarizes these in a *review issues list* to serve as an action list for the producer.
- A *formal technical review summary report* is created that answers three questions:
  1. What was reviewed?
  2. Who reviewed it?
  3. What were the findings and conclusions?
- You should establish a follow-up procedure to ensure that items on the issues list have been properly corrected.

Review Guidelines

- Review the product, not the producer.
- Set an agenda and maintain it.
- Limit debate and rebuttal.
- Enunciate problem areas, but don’t try 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.

Postmortem Evaluations

- A ***postmortem evaluation*** (PME) is a mechanism to determine what went right and what went wrong with the software engineering process and practices applied to a specific project.
- A PME is attended by members of the software team and stakeholders who examine the entire software project, focusing on *excellences* (achievements and positive experiences) and *challenges* (problems and negative experiences).
- The intent is to extract lessons learned from the challenges and excellences and to suggest improvements to process and practice moving forward.

Agile Reviews

- During the sprint planning meeting, user stories are reviewed and ordered according to priority.
- The daily Scrum meeting is an informal way to ensure that team members are all working on the same priorities and try to catch any defects that may cause the sprint to fail.
- The sprint review meeting is often conducted using guidelines like the formal technical review discussed in this chapter.
- The sprint retrospective meeting really a postmortem meeting in that the development team is trying to capture its lessons learned.

Software Quality Assurance

Quality Management

- Phil Crosby once said:
- The problem of quality management is not what people don't know about it. The problem is what they think they do.
- *Everybody is for it.* (Under certain conditions, of course.)
- *Everyone feels they understand it.* (Even though they wouldn't want to explain it.)
- *Everyone thinks execution is only a matter of following natural inclinations.* (After all, we do get along somehow.)
- *Most people feel that problems in these areas are caused by other people.* (If only they would take the time to do things right.)

Software Quality Assurance

- Access the text alternative for slide images.

Elements of SQA

- Standards.
- Reviews and Audits.
- Testing.
- Error/defect collection and analysis.
- Change management.
- Education.
- Vendor management.
- Security management.
- Safety.
- Risk management.

Data Driven SQA

- Access the text alternative for slide images.

Role of SQA Group 1

- **Prepares an SQA plan for a project which identifies:**
- Evaluations to be performed.
- Audits and reviews to be performed.
- Standards that are applicable to the project.
- Procedures for error reporting and tracking.
- Documents to be produced by the SQA group.
- Amount of feedback provided to the software project team.

- **Participates in the development of the project's software process description.**
- Reviews the process description for compliance with organizational policy, internal software standards, externally imposed standards (for example, ISO-9001), and other parts of the software project plan.

Role of SQA Group 2

- **Reviews software engineering activities to verify compliance with the defined software process.**
- Identifies, documents, and tracks deviations from the process and verifies that corrections have been made.

- **Audits designated software work products to verify compliance with those defined as part of the software process.**
- Reviews selected work products; identifies, documents, and tracks deviations; verifies that corrections have been made.
- Periodically reports the results of its work to the project manager.

- **Ensures that deviations in software work and work products are documented and handled according to a documented procedure.**
- **Records any noncompliance and reports to senior management.**
- Noncompliance items are tracked until they are resolved.

SQA Goals

- **Requirements quality.** The correctness, completeness, and consistency of the requirements model will have a strong influence on the quality of all work products.
- **Design quality.** Every element of the design model should be assessed to ensure that it exhibits high quality and that the design itself conforms to requirements.
- **Code quality.** Source code and related work products must conform to local coding standards and exhibit characteristics that will facilitate maintainability.
- **Quality control effectiveness.** A software team should apply limited resources in a way that has the highest likelihood of achieving a high quality result.

Formal SQA

- Assumes that a rigorous syntax and semantics can be defined for every programming language.
- Allows the use of a rigorous approach to the specification of software requirements.
- Applies mathematical proof of correctness techniques to demonstrate that a program conforms to its specification.
- Although formal methods are interesting to some software engineering researchers, most commercial developers rarely use of formal methods.

Statistical SQA

1. Information about software errors and defects is collected and categorized.
2. An attempt is made to trace each error and defect to its underlying cause (for example, design error, violation of standards, non-conformance to specifications, poor communication with the customer).
3. Using the Pareto principle (80 percent of the defects can be traced to 20 percent of all possible causes), isolate the 20 percent (the *vital few*).
4. Once the vital few causes have been identified, move to correct the problems that caused the errors and defects.

Six Sigma for Software Engineering 1

The term “six sigma” is derived from six standard deviations from the mean - 3.4 instances (defects) per million occurrences - implying an extremely high quality standard.

The three cores steps:

- ***Define*** customer requirements and deliverables and project goals via well-defined methods of customer communication.
- ***Measure*** the existing process and its output to determine current quality performance (collect defect metrics).
- ***Analyze*** defect metrics and determine the vital few causes.

Six Sigma for Software Engineering 2

For an existing process the needs improvement:

- ***Improve*** the process by eliminating the root causes of defects.
- ***Control*** the process to ensure that future work does not reintroduce the causes of defects.

For a new process being developed:

- ***Design*** the process to: (1) avoid the root causes of defects and (2) to meet customer requirements.
- ***Verify*** that the process model will, in fact, avoid defects and meet customer requirements.

Software Reliability and Availability

- A simple measure of reliability is *mean-time-between-failure* (MTBF):
  - $$MTBF = MTTF + MTTR$$
- MTTF is *mean-time-to-failure*
- MTTR is *mean-time-to-repair*, respectively.
- *Software availability* is the probability that a program is operating according to requirements at a given point in time and is defined as
  - $$Availability = [MTTF/(MTTF + MTTR)] \times 100\%$$

AI and Reliability Models

- ***Software reliability*** is the probability of failure-free software operation for a specified time period in a specified environment.
- ***Bayesian inference*** is a method of statistical inference in which Bayes' theorem is used to update the probability for a hypothesis (such as system reliability) being correct as more evidence or information becomes available.
- Bayesian inference can be used to estimate probabilistic quantities using historic data even when some of the information is missing.
- Making use of predictive data analytics tools such as a ***regression model*** involving MTBF can been used to

Software Safety

- ***Software safety*** is a software quality assurance activity that focuses on the identification and assessment of potential hazards that may affect software negatively and cause an entire system to fail.

- If hazards can be identified early in the software process, software design features can be specified that will either eliminate or control potential hazards.

ISO 9001:2015 Standard

- ISO 9001:2015 is the quality assurance standard that applies to software engineering.
- The requirements delineated by ISO 9001:2008 address topics such as:

  Management responsibility, quality system, contract review, design control, document and data control, product identification and traceability, process control, inspection and testing, corrective and preventive action, control of quality records, internal quality audits, training, servicing, and statistical techniques.

- For an organization to become registered to ISO 9001:2015, it must establish procedures to address each of the requirements listed and able to demonstrate these policies and being followed.

SQA Plan Contents

1. Purpose and scope of the plan.
2. Description of all software engineering work products that fall within the purview of SQA.
3. Applicable standards and practices that are applied during the software process.
4. SQA actions and tasks (including reviews and audits) and their placement throughout the software process.
5. Tools and methods supporting SQA actions and tasks.
6. Software configuration management procedures.
7. Methods for safeguarding and maintaining SQA records.
8. Organizational roles and responsibilities.