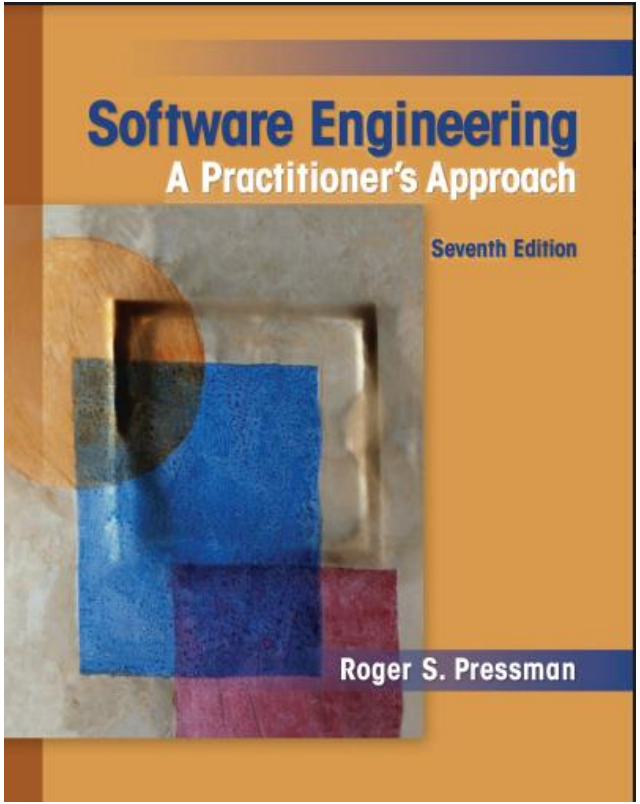


Prof. Dr. Gouri M. Patil ***(Asst. Professor)***
(Ph.D, M.Phil.(CS), MSc(CS), MCM)
BAS&PONahata Commerse College, Bhusawal

Unit- 6 Quality Management

- ✓ What is quality?
- ✓ Software quality
- ✓ Gravin’s quality dimension
- ✓ McCall’s quality factor
- ✓ ISO 9126 quality factors
- ✓ Targeted quality factor
- ✓ Review Technique- Formal Technical Review

Software Quality

- **Definition:**

  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 serves to emphasize three important points:**

  1. An effective software process establishes the infrastructure that supports any effort at building a high-quality software product.
  2. *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.
  3. 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.

1. *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.

2. *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.

3. 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. This enables software engineers to spend more time creating new applications and less on rework. 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.

Garvin’s Quality Dimensions

- Although Garvin’s eight dimensions of quality were not developed specifically for software, they can be applied when soft-ware quality is considered:

✓ **Performance quality**

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?

✓ **Feature quality**

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?

✓ **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?

Garvin's Quality Dimensions Cont…..

- ✓ **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? Douglas Adams [Ada93] makes a wry comment that seems appropriate here: "The difference between something that can go wrong and something that can't possibly go wrong is that when something that can't possibly go wrong goes wrong it usually turns out to be impossible to get at or repair."

- ✓ **Aesthetics**

  There's no question that each of us has a different and very subjective vision of what is aesthetic. And yet, 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 are evident nonetheless. Aesthetic software has these characteristics.

- ✓ **Perception**

  In some situations, you have a set of prejudices that will influence your perception of quality. For example, if you are introduced to a software product that was built by a vendor who has produced poor quality in the past, your guard will be raised and your perception of the current software product quality might be influenced negatively. Similarly, if a vendor has an excellent reputation, you may perceive quality, even when it does not really exist.

Garvin's Quality Dimensions Cont…..

- Garvin's quality dimensions provide you with a "soft" look at software quality.
- Many (but not all) of these dimensions can only be considered subjectively.
- For this reason, you also need a set of "hard" quality factors that can be categorized in two broad groups:

  (1) factors that can be directly measured (e.g., defects uncovered during testing) and

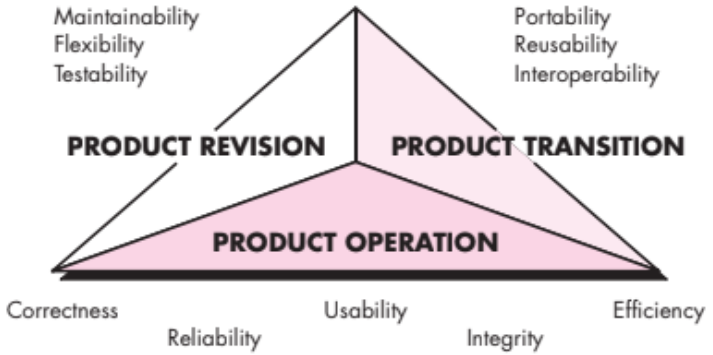
  (2) factors that can be measured only indirectly (e.g., usability or maintainability).

McCall's Quality Factors

McCall, Richards, and Walters propose a useful categorization of factors that affect software quality.

These software quality factors, shown in Figure focus on three important aspects of a software product:

- ✓ its operational characteristics
- ✓ its ability to undergo change and
- ✓ its adaptability to new environments



McCall's Quality Factors Cont….

1. **Correctness.** The extent to which a program satisfies its specification and fulfills the customer's mission objectives.
2. **Reliability.** The extent to which a program can be expected to perform its intended function with required precision. [It should be noted that other, more complete definitions of reliability have been proposed]
3. **Efficiency.** The amount of computing resources and code required by a program to perform its function.
4. **Integrity.** Extent to which access to software or data by unauthorized persons can be controlled.
5. **Usability.** Effort required to learn, operate, prepare input for, and interpret output of a program.
6. **Maintainability.** Effort required to locate and fix an error in a program. [This is a very limited definition.]
7. **Flexibility.** Effort required to modify an operational program.

McCall's Quality Factors Cont….

8. **Flexibility.** Effort required to modify an operational program.
9. **Testability.** Effort required to test a program to ensure that it performs its intended function.
10. **Portability.** Effort required to transfer the program from one hardware and/or software system environment to another.
11. **Reusability.** Extent to which a program [or parts of a program] can be reused in other applications related to the packaging and scope of the functions that the program performs.
12. **Interoperability.** Effort required to couple one system to another.

- It is difficult, and in some cases impossible, to develop direct measures2 of these quality factors. In fact, many of the metrics defined by McCall et al. can be measured only indirectly. However, assessing the quality of an application using these factors will provide you with a solid indication of software quality.

ISO 9126 Quality Factors

The ISO 9126 standard was developed in an attempt to identify the key quality attributes for computer software.

The standard identifies six key quality attributes:

1. **Functionality.** *The degree to which the software satisfies stated needs* as indicated by sub-attributes: suitability, accuracy, interoperability, compliance, and security.
2. **Reliability.** *The amount of time that the software is available for use* as indicated by sub-attributes: maturity, fault tolerance, recoverability.
3. **Usability.** *The degree to which the software is easy to use* as indicated by sub-attributes: understandability, learnability, operability.
4. **Efficiency.** *The degree to which the software makes optimal use of system resources* as indicated by sub-attributes: time behavior, resource behavior.
5. **Maintainability.** *The ease with which repair may be made to the software* as indicated by sub-attributes: analyzability, changeability, stability, testability.
6. **Portability.** *The ease with which the software can be transposed from one environment to another* as indicated by sub-attributes: adaptability, installability, conformance, replaceability.

- Like other software quality factors, the ISO 9126 factors do not necessarily provide themselves to direct measurement.
- However, they do provide a worthwhile basis for indirect measures and an excellent checklist for assessing the quality of a system.

Targeted Quality Factors

- ❑ McCall identifies usability as an important quality factor.
- ❑ In a pragmatic sense, to conduct the assessment, one will need to address specific, measurable (or at least, recognizable) attributes of the interface.

For Example:

- ✓ **Intuitiveness.** The degree to which the interface follows expected usage patterns so that even a novice can use it without significant training.
  - Is the interface layout conducive to easy understanding?
  - Are interface operations easy to locate and initiate?
  - Does the interface use a recognizable metaphor?
  - Is input specified to economize key strokes or mouse clicks?
  - Does the interface follow the three golden rules? (Chapter 11)
  - Do aesthetics aid in understanding and usage?

Targeted Quality Factors Cont …

- ✓ **Efficiency.** The degree to which operations and information can be located or initiated.
  - Does the interface layout and style allow a user to locate operations and information efficiently?
  - Can a sequence of operations (or data input) be performed with an economy of motion?
  - Are output data or content presented so that it is understood immediately?
  - Have hierarchical operations been organized in a way that minimizes the depth to which a user must navigate to get something done?
- ✓ **Robustness.** The degree to which the software handles bad input data or inappropriate user interaction.
  - Will the software recognize the error if data at or just outside prescribed boundaries is input? More importantly, will the software continue to operate without failure or degradation?
  - Will the interface recognize common cognitive or manipulative mistakes and explicitly guide the user back on the right track?
  - Does the interface provide useful diagnosis and guidance when an error condition (associated with software functionality) is uncovered?

Targeted Quality Factors Cont …

- ✓ **Richness.** The degree to which the interface provides a rich feature set.
  - Can the interface be customized to the specific needs of a user?
  - Does the interface provide a macro capability that enables a user to identify a sequence of common operations with a single action or command?

Note: *As the interface design is developed, the software team would review the design prototype and ask the questions noted. If the answer to most of these questions is "yes," it is likely that the user interface exhibits high quality. A collection of questions similar to these would be developed for each quality factor to be assessed.*

Review Technique- Formal Technical Review

- ***Software reviews*** are a “filter” for the software process. That is, reviews are applied at various points during software engineering and serve to uncover errors and defects that can then be removed. Software reviews “purify” software engineering work products, including requirements and design models, code, and testing data.
- ***A review*** (any review) is a way to:

  1. Point out needed improvements in the product of a single person or team;
  2. Confirm those parts of a product in which improvement is either not desired or not needed;
  3. Achieve technical work of more uniform, or at least more predictable, quality than can be achieved without reviews, in order to make technical work more manageable.

Types of Reviews

- ***Informal reviews*** are characterized by minimal planning and preparation and little record keeping. Desk checks and pair programming fall into the informal review category.

- ***A Formal Technical Review (FTR)*** is a stylized meeting that has been shown to be extremely effective in uncovering errors. Walkthroughs and inspections establish defined roles for each reviewer, encourage planning and advance preparation, require the application of defined review guidelines, and mandate record keeping and status reporting. Sample-driven reviews can be used when it is not possible to conduct formal technical reviews for all work products.

Formal Technical Review (FTR) Cont …

A formal technical review (FTR) is a software quality control activity performed by software engineers (and others).

The objectives of an FTR 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; and
(5) to make projects more manageable.

In addition, the FTR serves as a training ground, enabling junior engineers to observe different approaches to software analysis, design, and implementation.

- The FTR also serves to promote backup and continuity because a number of people become familiar with parts of the software that they may not have otherwise seen.

- **The FTR is actually a class of reviews that includes walkthroughs and inspections.**
- **Each FTR is conducted as a meeting and will be successful only if it is properly planned, controlled, and attended.**

❖ The Review Meeting:

Every review meeting should abide by the following constraints:

- ❑ 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. Given these constraints, it should be obvious that an FTR focuses on a specific (and small) part of the overall software.
- ❑ For example, rather than attempting to review an entire design, walkthroughs are conducted for each component or small group of components.

❖ Review Summary Report

- ❑ What was reviewed?
- ❑ Who reviewed it?
- ❑ What were the findings and conclusions?

❖ The Players of Review Meeting

- **Producer**—the individual who has developed the work product

  Informs the project leader that the work product is complete and that a review is required.

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


Wishing
You
The Best Of Luck!!