SOFTWARE ENGINEERING

Presented by
Pranalini

Unit 5 Part 2: QUALITY MANAGEMENT

Index:

Software Quality

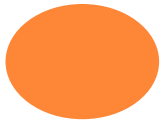
Informal Reviews

Formal Technical Reviews

Statistical Software Quality Assurance

Software Reliability

The ISO 9000 Quality Standards



QUALITY

- **OED(Oxford English Dictionary)**, 1990

Degree of excellence of an item.

- **CROSSBY, 1979**

Product with Zero defects.

- **ISO, 1986**

The totally of feature and characteristics of a product/Service that bear on its ability to satisfied/implied needs.

- Quality is defined as the product/service capability to meet the **customer expectations** and there by provides customer to the **product satisfaction.**
- Quality that is defined as a matter of products and service whose measurable characteristics satisfied a **fixed specification**.
- Quality is defined as a **conformance to requirements**.

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

- In nutshell, Quality is a defined as a Characteristics and attributes of something where as attribute refer to measurable characteristics- things that we are able to **compare to known standards.**

QUALITY TYPES

QUALITY OF DESIGN

- It refers to the characteristic that designer specify for an item. The grade of material, tolerance and performance specification all contribute to the quality of design.

QUALITY OF CONFORMANCE

- It is the degree to which design specification are followed during manufacturing. Greater the degree of conformance, the higher is the level of quality of conformance.

SOFTWARE QUALITY

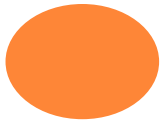
- **Software Quality** is An effective software process applied in a manner that creates a useful product that provides measurable value for those who produce it and those who use it.

- **Software Quality in project management,** Quality will be of concern at all stages of project planning and execution. But will be of particular interest at the following points in the step wise framework

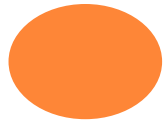
1. An *effective software process establishes the infrastructure that supports any* effort at building a high-quality software product
2. A *useful product delivers the content, functions, and features that the end* user desires, but as important, it delivers these assets in a reliable, error-freeway.
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.

GARVIN'S QUALITY DIMENSIONS

- David Garvin suggests that quality should be considered by taking a multidimensional viewpoint that **begins with an assessment of conformance and terminates with a transcendental (aesthetic) view**. Although Garvin's eight dimensions of quality were not developed specifically for software, they can be applied when software quality is considered

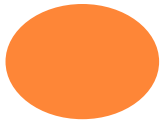


- **Performance Quality**. Does the software deliver all content, functions, and features that are specified as part of the requirements model in a way that provides value to the end user?
- **Feature quality**. Does the software provide features that surprise and delight first-time end users?
- **Reliability.** 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?



- **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?

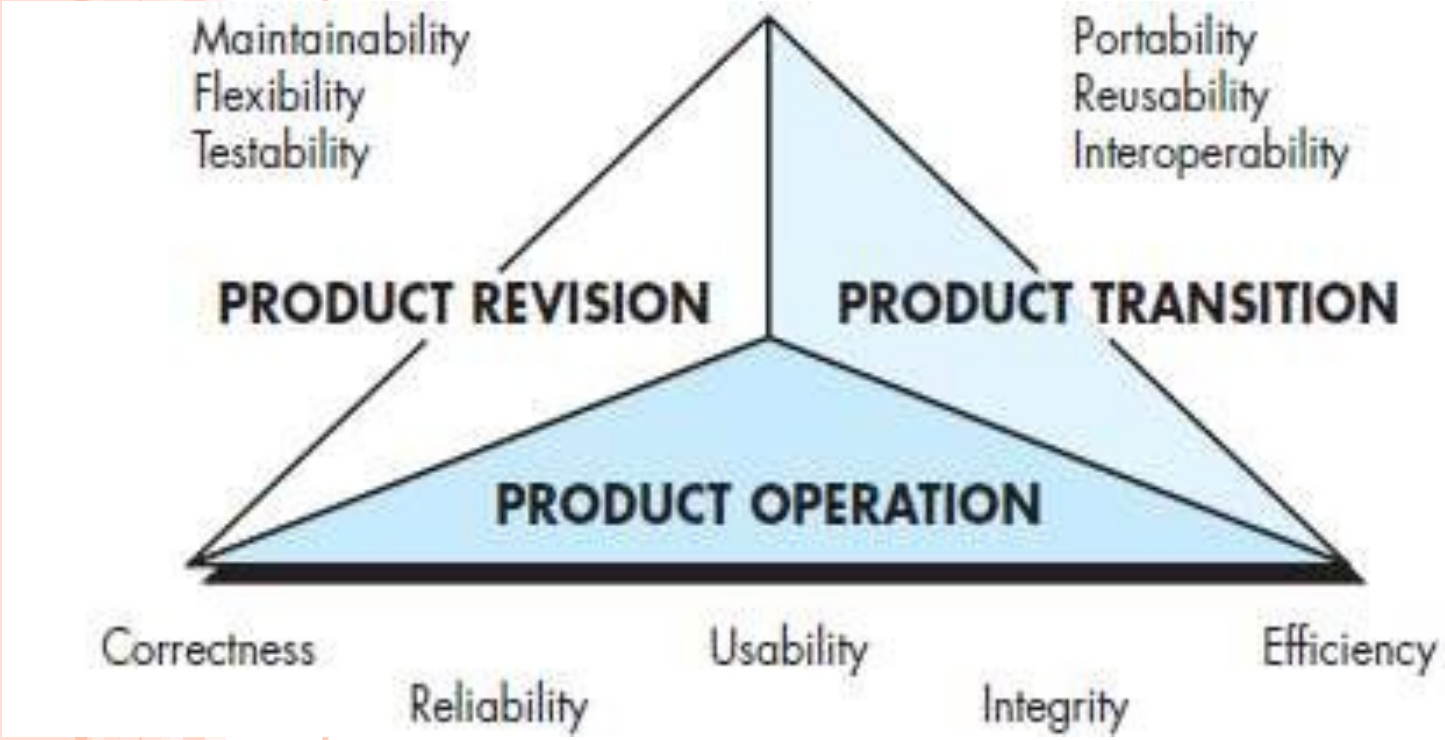


- **Aesthetics**. An aesthetic entity has a certain elegance, a unique flow, and an obvious "presence" that are hard to quantify but are evident nonetheless.

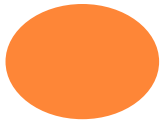
- **Perception**. In some situations, you have a set of prejudices that will influence your perception of quality. For example, if you are introduced to a software product that was built by a vendor who has produced poor quality in the past, your guard will be raised and your perception of the current 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.

MCCALL'S QUALITY FACTORS

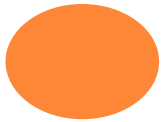
Software quality factors focus on three important aspects of a software product: its operational characteristics, its ability to undergo change, and its adaptability to new environments.



- **Correctness**. The extent to which a program satisfies its specification and fulfills the customer's mission objectives.
- **Reliability**. The extent to which a program can be expected to perform its intended function with required precision.
- **Efficiency**. The amount of computing resources and code required by a program to perform its function.
- **Integrity**. Extent to which access to software or data by unauthorized persons can be controlled.

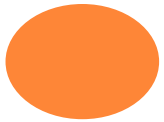


- **Usability**. Effort required to learn, operate, prepare input for, and interpret output of a program.
- **Maintainability**. Effort required to locate and fix an error in a program.
- **Flexibility**. Effort required to modify an operational program.
- **Testability**. Effort required to test a program to ensure that it performs its intended function.
    - **Portability**. Effort required to transfer the program from one hardware and/or software system environment to another.



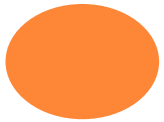
- **Reusability**. Extent to which a program can be reused in other applications - related to the packaging and scope of the functions that the program performs.

- **Interoperability**. Effort required to couple one system to another.



ISO 9126 QUALITY FACTORS

- **Functionality.** The degree to which the software satisfies stated needs as indicated by the following sub attributes: suitability, accuracy, interoperability, compliance, and security.
- **Reliability.** The amount of time that the software is available for use as indicated by the following sub attributes: maturity, fault tolerance, recoverability.
- **Usability** . The degree to which the software is easy to use as indicated by the following sub attributes: understandability, learnability, operability.



- **Efficiency.** The degree to which the software makes optimal use of system resources as indicated by the following sub attributes: time behavior, resource behavior.
- **Maintainability.** The ease with which repair may be made to the software as indicated by the following sub attributes: analyzability, changeability, stability, testability.
- **Portability.** The ease with which the software can be transposed from one environment to another as indicated by the following sub attributes: adaptability, installability, conformance, replaceability.

TARGETED QUALITY FACTORS

Intuitiveness. The degree to which the interface follows expected usage patterns so that even a novice can use it without significant training.

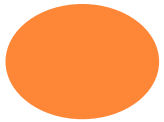
(1). Are interface operations easy to locate and initiate?

(2). Does the interface use a recognizable metaphor?

(3). Is input specified to economize key strokes or mouse clicks?

(4). Does the interface follow the golden rules?

(5). Do aesthetics aid in understanding and usage?

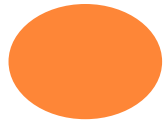


Efficiency. The degree to which operations and information can be located or initiated.

- 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?

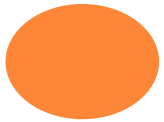
Robustness. The degree to which the software handles bad input data or inappropriate user interaction.

- Will the software recognize the error if data values are at or just outside prescribed input boundaries? 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?



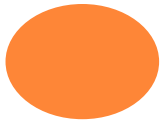
Richness. The degree to which the interface provides a rich feature set.

- 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?
- 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.



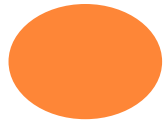
The Transition to a Quantitative View

- Till now we have discussed variety of qualitative factors for the "measurement" of software quality.
- The software engineering community strives to develop precise measures for software quality and is sometimes frustrated by the subjective nature of the activity.
- In all cases, the metrics represent indirect measures; that is, we never really measure *quality but rather some manifestation of* quality.
- The complicating factor is the precise relationship between the variable that is measured and the quality of software.



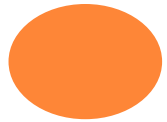
Informal Reviews

- 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

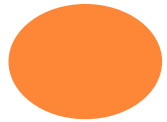
- The objectives of an FTR are:
  - to uncover errors in function, logic, or implementation for any representation of the software
  - to verify that the software under review meets its requirements
  - to ensure that the software has been represented according to predefined standards
  - to achieve 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



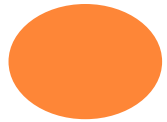
The Review Meeting

1. Between three and five people (typically) should be involved in the review.
2. Advance preparation should occur but should require no more than two hours of work for each person.
3. The duration of the review meeting should be less than two hours.

- Focus is on a work product (e.g., a portion of a requirements model, a detailed component design, source code for a component)
- The individual who has developed the work product i.e, the producer informs the project leader that the work product is complete and that a review is required.
- The project leader contacts a review leader, who evaluates the product for readiness, generates copy of product material and distributes them to two or three review members for advance preparation .
- Each reviewer is expected to spend between one and two hours reviewing the product, making notes



- The review leader also reviews the product and establish an agenda for the review meeting
- The review meeting is attended by review leader, all reviewers and the producer.
- One of the reviewer act as a recorder, who notes down all important points discussed in the meeting.
- The meeting(FTR) is started by introducing the agenda of meeting and then the producer introduces his product. Then the producer “walkthrough” the product, the reviewers raise issues which they have prepared in advance.
- If errors are found the recorder notes down



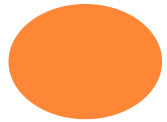
Review reporting and Record keeping

- During the FTR, a reviewer(recorder) records all issues that have been raised
- A review summary report answers three questions

1. What was reviewed?
2. Who reviewed it?
3. What were the findings and conclusions?

- Review summary report is a single page form with possible attachments
- The review issues list serves two purposes

1. To identify problem areas in the product
2. To serve as an action item checklist that guides the producer as corrections are made

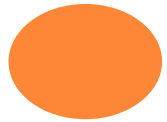


Review Guidelines

- 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 i.e likely to be reviewed
- Allocate resources and schedule time for FTRS
- Conduct meaningful training for all reviewer
- Review your early reviews

Sample Driven Reviews

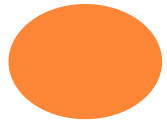
- In real world of s/w projects, resources are limited and time is short.
- In such situations, reviews are often skipped.
- **Thelin and his colleagues** address this issue by suggesting a sample-driven review process.
- In this, samples of all s/w work products are inspected to determine which work products are more error prone.
- Full FTR resources are then focused only to those work products that are likely to be error-prone .



- Sample driven review must attempt to quantify those work products that are primary targets for full FTRs.
- Following are the suggested steps:
  - Inspect a fraction a_i of each software work product, i. Record the number of faults, f_i found within a_i
  - Develop a gross estimate of the no. of faults within the work product i by multiplying f_i by $1/ a_i$
  - Sort the work products in descending order according to the gross estimate of the number of faults in each.
  - Focus available review resources on those work products that have the highest estimated number of faults
- The fraction of work product that is sampled must be
  - …representative of the work product as a whole and
  - …Large enough to be meaningful to the reviewer(s) who does the sampling

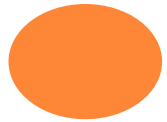
Statistical Software Quality Assurance

- It's a quantitative approach about quality. Following are the steps:
  1. Information about software defects is collected and categorized
  2. An attempt is made to trace each defect to its underlying cause (e.g., non-conformance to its specification, design error, violation of standards, etc.)
  3. Using the Pareto Principle (80% of the defects can be traced to 20% of all possible causes), isolate the 20% "vital few"
  4. Once the vital few, move to correct the problems that have caused the defects.



A Generic Example

- Software engineering organization collects information on errors and defects for a period of one year. Some of the errors are uncovered as software is being developed.
- Others (defects) are encountered after the software has been released to its end users. Although hundreds of different problems are uncovered, all can be tracked to one (or more) of the following causes:
  - Incomplete or erroneous specifications (IES)
  - Misinterpretation of customer communication (MCC)
  - Intentional deviation from specifications (IDS)
  - Violation of programming standards (VPS)



- Error in data representation (EDR)
- Inconsistent component interface (ICI)
- Error in design logic (EDL)
- Incomplete or erroneous testing (IET)
- Inaccurate or incomplete documentation (IID)
- Error in programming language translation of design (PLT)
- Ambiguous or inconsistent human/computer interface (HCI)
- Miscellaneous (MIS)

- To apply statistical SQA. The table indicates that IES, MCC, and EDR are the vital few causes that account for 53 percent of all errors. It should be noted, however, that IES, EDR, PLT, and EDL would be selected as the vital few causes if only serious errors are considered.

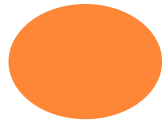
Data collection for statistical SQA

	Total		Serious		Moderate		Minor	
Error	No.	%	No.	%	No.	%	No.	%
IES	205	22%	34	27%	68	18%	103	24%
MCC	156	17%	12	9%	68	18%	76	17%
IDS	48	5%	1	1%	24	6%	23	5%
VPS	25	3%	0	0%	15	4%	10	2%
EDR	130	14%	26	20%	68	18%	36	8%
ICI	58	6%	9	7%	18	5%	31	7%
EDL	45	5%	14	11%	12	3%	19	4%
IET	95	10%	12	9%	35	9%	48	11%
IID	36	4%	2	2%	20	5%	14	3%
PLT	60	6%	15	12%	19	5%	26	6%
HCI	28	3%	3	2%	17	4%	8	2%
MIS	56	6%	0	0%	15	4%	41	9%
Totals	942	100%	128	100%	379	100%	435	100%

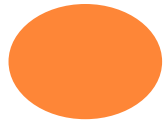
Six Sigma

- It is the most widely used strategy used in industry for statistical quality assurance.
- It was originally popularized by Motorola in 1980s.
- It can be described as
  - A rigorous and disciplined methodology that uses data and the statistical analysis to measure & improve a company's operational performance by identifying and eliminating 'defects' in manufacturing & service-related processes

- Six sigma methodology defines 3 core steps:
  - **Define** customer requirements, deliverables, & project goals via well defined methods of customer comm.
  - **Measure** the existing process & its output to determine current quality performance (collect defect metrics)
  - **Analyze** defect metrics & determine the vital few causes.
- If an existing software process is in place, but improvement is required, Six Sigma suggests 2 additional steps:
  - **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
- These core steps and additional steps are also referred to as **DMAIC** method

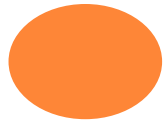


- If an organization is developing a software process (rather than improving an existing one), the core steps are augmented by:
  - **Design** the process to (1) avoid the root causes of defects and (2) to meet the customer requirements
  - **Verify** that the process model will, in fact, avoid defects and meet customer requirements
- This is referred to as **DMADV** method



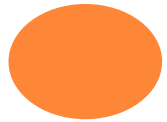
Software Reliability

- Defined as the probability of failure free operation of a computer program in a specified environment for a specified time period
- Can be measured directly and estimated using historical and developmental data
- Software reliability problems can usually be traced back to errors in design or implementation.
- What is meant by the term *failure?*
  - failure is nonconformance to software requirements



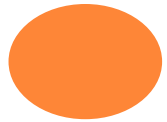
Measures of Reliability and Availability

- In hardware, failures due to physical wear (e.g., the effects of temperature, corrosion, shock) are more likely than a design-related failure.
- Software failures can be traced to design or implementation problems; wear does not enter into the picture.
- Measures of Reliability
  - Mean time between failure (MTBF) = MTTF + MTTR
  - MTTF = mean time to failure
  - MTTR = mean time to repair
  - Availability = [MTTF / (MTTF + MTTR)] x 100%



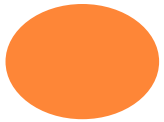
Software Safety

- *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.
- Some of the hazards associated with a computer-based cruise control for an automobile might be:
  (1) causes uncontrolled acceleration that cannot be stopped,
  (2) does not respond to depression of brake pedal (by turning off),
  (3) does not engage when switch is activated, and
  (4) slowly loses or gains speed.
- Once these system-level hazards are identified, analysis techniques are used to assign severity and probability of occurrence
- To be effective, software must be analyzed in the context of the entire system.



ISO 9000 Quality Standards

- Quality assurance systems are defined as the organizational structure, responsibilities, procedures, processes, and resources for implementing quality management.
- ISO 9000 describes the quality elements that must be present for a quality assurance system to be compliant with the standard, but it does not describe how an organization should implement these elements.
- ISO 9001:2000 is the quality standard that contains 20 requirements that must be present in an effective software quality assurance system.



SUMMARY

- Quality is defined as the product/service capability to meet the customer expectations and there by provides customer to the product satisfaction.
- 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.
- David Garvin suggests that quality should be considered by taking a multidimensional viewpoint that begins with an assessment of conformance and terminates with a transcendental (aesthetic) view.
- Software quality factors focus on three important aspects of a software product: its operational characteristics, its ability to undergo change, and its adaptability to new environments.
- 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.
- The cost of quality depends on three terms: Prevention, Appraisal, Failure cost

THANK YOU!