Software Quality Engineering

Chapter 1 - Software Quality Fundamentals

Created by Dani Budinova-Spahieva

Press **ESC** to enter the slide overview.
Use the *Space* key to navigate through all slides.
Presentations look great on touch devices, like mobile phones and tablets. Simply swipe through your slides.

1.1 Software Engineering Culture and Ethics

The culture of an organization is a critical success factor in its process improvement ef- forts. "Culture" includes a set of shared values and principles that guide the behaviors, activities, priorities, and decisions of a group of people working in the same area. When coworkers align along common beliefs, it is easier to induce changes that will increase the group's effectiveness and its probability of survival. A shared culture is one difference between a "team" and a "bunch of bozos on a bus.

Customer involvement is the most critical factor in software quality.

Any successful enterprise requires that you both do the right thing, and do the thing right. We ex- pect that professional software developers know how to do the "thing" right. Doing the right thing, though, requires an unambiguous understanding of what your customer expects. To this end, we strive to maximize participation of our customers in our development activities.

Sharing the vision of the final product with the customer

The classic failure of software development is that the product delivered only vaguely matches the expectations of the customer. Every project, even small ones developing internally reusable software components, goes through a formal, written requirements specification process If you don't have time to do it right, when will you have time to do it over?

Quality is the top priority; long-term productivity is a natural consequence of high quality

Ongoing education is every team member's responsibility.

Have a peer, rather than a customer, find a defect.

Written software development procedures can help improve quality.

Continual improvement of your software development process is both possible and essential.

Never let your boss or your customer talk you into doing a bad job.

People need to feel that the work they do is noticed.

Summary

You can't buy a culture in shrink-wrap; you must roll your own. Every software team works in a different context of expectations, pressures, application domains, and technolo- gies. The process of agreeing upon principles and values provides many opportunities for im- proving both the work environment and the work results. A shared culture is essential to progress through the software process maturity sequence to the discipline of repeatable and measurable software development processes.

Value and Cost of Quality (COQ)

Associated with providing poor quality products or services. Categories:

- **internal failure costs**

  associated with defects found before the customer receives the product or service

- **external failure costs**

  associated with defects found after the customer receives the product or service

- **appraisal costs**

  incurred to determine the degree of conformance to quality requirements

- **prevention costs**

  incurred to keep failure and appraisal costs to a minimum.

Cost of quality

is a methodology that allows an organization to determine the extent to which its resources are used for activities that prevent poor quality, that appraise the quality of the organization's products or services, and that result from internal and external failures. Having such information allows an organization to determine the potential savings to be gained by implementing process improvements.

Quality-related activities that incur costs may be divided into:

- **prevention costs**
- **appraisal costs**
- **internal costs**
- **external costs**

Prevention costs

Incurred to prevent or avoid quality problems. Associated with the design, implementation, and maintenance of the quality management system. Planned and incurred before actual operation, and they could include:

- **Product or service requirements**

  establishment of specifications for incoming materials, processes, finished products, and services

- **Quality planning**

  creation of plans for quality, reliability, operations, production, and inspection

- **Quality assurance**

  creation and maintenance of the quality system

- **Training**

  development, preparation, and maintenance of programs

Appraisal costs

Associated with measuring and monitoring activities related to quality. These costs are associated with the suppliers' and customers' evaluation of purchased materials, processes, products, and services to ensure that they conform to specifications. They could include:

- **Verification**

  checking of incoming material, process setup, and products against agreed specifications

- **Quality audits**

  confirmation that the quality system is functioning correctly

- **Supplier rating**

  assessment and approval of suppliers of products and services

Internal failure costs

Incurred to remedy defects discovered before the product or service is delivered to the customer. These costs occur when the results of work fail to reach design quality standards and are detected before they are transferred to the customer.

They could include:

- **Waste**

  performance of unnecessary work or holding of stock as a result of errors, poor organization, or communication

- **Rework**

- **Failure analysis**

External failure costs

Incurred to remedy defects discovered by customers. These costs occur when products or services that fail to reach design quality standards are not detected until after transfer to the customer. They could include:

- **Repairs and servicing**

- **Warranty claims**

- **Complaints**

  all work and costs associated with handling and servicing customers' complaints

Cost of quality and organizational objectives

The costs of doing a quality job, conducting quality improvements, and achieving goals must be carefully managed so that the long-term effect of quality on the organization is a desirable one. The quality cost system, once established, should become dynamic and have a positive impact on the achievement of the organization's mission, goals, and objectives.

Models and Quality Characteristics

The official ISO 9126 documentation software quality model identifies 6 main quality characteristics, namely:

- **Functionality**

  The software system is functioning, as specified,

- **Reliability**

  the capability of the system to maintain its service provision under defined conditions for defined periods of time. (fault tolerance - the ability of a system to withstand component failure)

Models and Quality Characteristics - part 2

- **Usability**

  exists with regard to functionality and refers to the ease of use for a given function. learnability is also a major subcharacteristic of usability.

- **Efficiency**

  concerned with the system resources used when providing the required functionality.

- **Maintainability**

  ability to identify and fix a fault within a software component; testability, readability, modularization

- **Portability**

  how well the software can adopt to changes in its environment or with its requirements

Functionality

- **Suitability**

  appropriateness of the functions of the software.

- **Accurateness**

  correctness of the functions

- **Interoperability**

  the ability of a software component to interact with other components or systems.

- **Compliance**

  Where appropriate certain industry (or government) laws and guidelines need to be complied with

- **Security**

  unauthorized access to the software functions

Reliability

- **Maturity**

  frequency of failure of the software.

- **Fault tolerance**

  The ability of software to withstand (and recover) from component, or environmental, failure

- **Recoverability**

  Ability to bring back a failed system to full operation, including data and network connections.

Usability

Understandability

Determines the ease of which the systems functions can be understood

Operability

Ability of the software to be easily operated by a given user in a given environment.

Learnability

Learning effort for different users, i.e. novice, expert, casual etc.

Efficiency

Time behavior

Characterizes response times for a given thru put, i.e. transaction rate.

Resource behavior

Characterizes resources used, i.e. memory, cpu, disk and network usage.

Maintainability

Analyzability

Characterizes the ability to identify the root cause of a failure within the software

Changeability

Characterizes the amount of effort to change a system.

Stability

Characterizes the sensitivity to change of a given system that is the negative impact that may be caused by system changes.

Testability

Characterizes the effort needed to verify (test) a system change.

Portability

Adaptability

Characterizes the ability of the system to change to new specifications or operating environments.

Installability

Characterizes the effort required to install the software

Conformance

portability

Replaceability

Characterizes the *plug and play* aspect of software components, that is how easy is it to exchange a given software component within a specified environment.

Software Development Models

Testing does not exist in isolation, test activities are related to software development activities. Different development life cycle models need different approaches to testing. There are various Software development models or methodologies. http://istqbexamcertification.com/what-are-the-software-development-models/

Software Development Models:

- **Waterfall model**
- **V model**
- **Incremental model**
- **RAD model**
- **Agile model**
- **Iterative model**
- **Spiral model**
- **Prototype model**

Software Quality Improvement

Software underpins nearly every business process. Therefore it's essential to lower the cost of testing and improve the quality of your software. These four pointers will help you improve software quality and improve testing efficiency.

Test at the right time

By testing earlier you will be able to detect and solve defects rather than having to resolve them at the end of the process. The later software bugs are detected, the longer and more expensive they are to resolve. Fixing bugs early can be a game-changer. The firm cut development time by 25 percent and bug fixing costs by 31 percent. Get testers involved during the requirements and design stage so they can help formulate a more effective test framework. More than 70 percent of software issues in a live environment can be traced back to poor requirements. Implement Static testing early in the life cycle to give immediate reactions on quality issues regarding your software development.

Improve testing organisation

Be professional in your testing or risk harming your business. Implement specific policies to direct a consistent approach like using repeatable industry standard testing processes and training testers within this framework. Remember: Use metrics. This helps keep track of where you are ensuring that you don't start before you're ready, and you finish when you're done to prevent wasted bandwidth.

Innovation leads to improvement

Write automated tests wherever possible to make your testing process more efficient. Make sure that your testing is targeted too. By using test design techniques and risk-based testing you can make sure that you will be using fewer, but more worthwhile tests.

Keep reviewing

You can have too much of a good thing. Just because certain methods have worked in the past doesn't mean they always will. Processes of evaluation and refactoring allow your testing team to maximise efficiency by reviewing what worked well. Implement a Root cause analysis process which distinguishes whether issues were a 'testing miss', a 'development miss' or a 'requirements or design miss'. This will help to identify areas for improvement throughout the whole software development process.

Software system safety

In software engineering, software system safety optimizes system safety in the design, development, use, and maintenance of software systems and their integration with safety-critical hardware systems in an operational environment. There are three major types of software failure. Not all of these can be rectified by the programmers. The three types are:

- **Software logic errors,**
- **Software support errors,**
- **Hardware failures**

Techniques

- **During preliminary design**
- **During Analysis/Requirements**
- **During the Software Design phase**
- **During the late design and coding phase**
- **Software verification and testing**

References

http://www.fmeainfocentre.com/handbooks/2005-01-0785.pdf - Main source for techniques

http://www.dcs.gla.ac.uk/~johnson/teaching/safety/reports/Clif_Ericson1.htm

http://www.msha.gov/TECHSUPP/ACC/sysafety/softsafe32.pdf

http://satc.gsfc.nasa.gov/assure/distasst.pdf

http://www.embedded.com/columns/technicalinsights/19201765?_requestid=471140

http://en.wikipedia.org/wiki/Failure_mode_and_effects_analysis

http://www.weibull.com/basics/fault-tree/

Goddard, Peter. Software FMEA Techniques, 2000