

TS5130 System Development Theory and Practice
Mark Lindquist
Unit 1, Discussion 2

January 3rd, 2009

High Quality Software (Pfleeger & Atlee, 1995)
by Mark Lindquist

Software Engineering consists of producing high quality software. It is not enough to produce software that works ok or is acceptable. High quality in software is a normal, non-functional, requirement in the production of good software. But what makes software high quality?

High quality software consists of a number of attributes and characteristics that distinguish good software from poorly built software.

- **Does as intended according to the customer requirements.** This should be the number one consideration of high quality software. If the product does not do what the customer stipulates then the product is of little value.
- **Easy to Understand:** The software product should be easy to understand and be user friendly which characterizes high quality software.
- **Robust:** The software should perform exceptionally under a wide variety of conditions.
- **Reliable:** Being reliable is a high quality trait. Reliability means fewer faults (a combination of one or more human errors) or failures (a combination of human error leading to deviations from the customer requirements). Finding faults early in the development life cycle is less expensive than trying to fix a problem later.
- **Available:** A completed software product should be available at any given point in time to be considered high quality.
- **Maintainable:** If the system must be maintained (not a turnkey system) then members of the software team must use flexibility in making changes as required.

As Garvin (1984) points out about different perspectives of high quality, he notes that there are five views.

- **Transcendental view:** Hard to visualize, this view, strives toward the ideal quality of a product.
- **The user view:** Measurements are used to understand overall product quality.
- **The manufacturing view:** Important during production and after delivery, this view determines whether the product was built right initially.
- **The product view:** This view takes an internal approach viewing the products inherent characteristics.
- **The value based view:** This view distinguishes among a customer's user view, a researcher's product view, and the development team's manufacturing view.

Kitchenham and Pfleeger (1996) thus define three areas to consider high quality software.

- **The quality of the product:** The quality of the product can be assessed in two ways. The users view of what the product is supposed to do and the practitioners who develop the product. Their view is internal and relates to the functions the product performs.
- **The quality of the process:** The Capability Maturity Model (CSM), ISO 9000 and Software Process Improvement and Capability dEtermination (SPICE) say that improving the process will lead to improved high quality software. The quality of development and maintenance processes will lead to higher quality.
- **Quality in the context of the business environment:** Technical quality of the product will improve the business value. Quantitative measurements are used to determine the relationship between technical value and business value.

Wasserman (1996) addresses the transformation that software engineering has taken as a result of technological changes. He describes eight notions that form the basis of effective software engineering as a result.

- **Abstraction:** To take an idea and transform it into a detailed functioning product can be difficult. Abstraction lets you examine a problem at some level of generalization leaving out the more complicated details.
- **Analysis and Design Methods and Notations:** Because there are no standard notations in software engineering, we use common methods and techniques to gain an understanding of a problem.
- **User Interface Prototyping:** A small, incomplete system, used for identifying key requirements of the system, and the feasibility of a system. The prototype is usually iterative meaning it is built, revised and built again. In larger systems the iterative nature is best done on sub-systems, then combining the sub-systems into the system itself. This is called phased development.
- **Software Architecture:** System architecture addresses five units and how they relate to one another; modular decomposition, data oriented decomposition, event oriented decomposition, outside-in-design, and object oriented design.
- **Software Process:** The actual process is known to produce quality software but different processes are prescribed for different parts of the project because no one process works for everything. Large projects need more “Control” than small projects.
- **Reuse:** Reuses is the process of using reusing components that are part of other applications. For instance an open button on a spreadsheet program can be used in another program.
- **Measurement:** A quantitative approach for improving a software product.
- **Tools and Integrated Environments:** Choosing the right environment or tool for the task requires examining features of each and deciding which is better suited for the development work.

As a final thought about people and the process of developing a product, Bernard (2003) suggests that people not technology are the cause of system failures. In fact if technology seems the likely culprit of a failure, then it is usually indirectly linked to a person. Technology as the cause of failures simply does not happen very much.

As a result he goes on further to explain that it is change management where the human failures lie. All projects should consist of a change management process to inevitably deal with changes the system will undoubtedly go through. In today's software creation process, change creeps in at all stages of development: the requirements analysis, system design, program design, implementation, testing, delivery and maintainability. Change happens throughout the development stages and must elicit a flexible process in dealing with the change.

Planning is also important in avoiding failures. An essential plan is necessary to avoid these failures. This is common sense. A well thought out plan involving the building of software is a better solution than not having a plan at all. Carrying out the plan is as important as the plan itself.

Training is also an important part of change management. To many times people don't have the training needed to effectively handle change.

My experience with software and software quality has to do with my websites that I have built for small businesses. After many years of starting a project not knowing the customers requirements it became clear that this one area of high quality is paramount to distinguishing good quality websites from bad ones.

In the actual creation of the websites the customers expectations (requirements) actually are essential in producing the "correct" website. Otherwise the website fails, it may look and work well, but failure is the result of omitting this quality attribute.

The website should also be "easy to understand", "robust", "reliable", "available" and "maintainable".

In conclusion software is good software when high quality is factored into the software product life cycle.

References:

Pfleeger, S.L. & Atlee, J.M. (2006). *Software Engineering Theory and Practice, Third Edition*. Upper Saddle River, NJ: Pearson prentice Hall.

Garvin, D. (1984). "What does product quality really mean?" Sloan Management Review, (fall): 25 – 45.

Kitchenham & Pfleeger, (1996). "Software Quality: The allusive target." IEEE Software, 13(1) (January): 12 – 21.

Wasserman, (1996). "Toward a discipline of software engineering." IEEE Software, 13(6)(November) 23 - 31

Bernard, A. (2003). "Why implementations fail, the human factor"

This document was created with Win2PDF available at <http://www.win2pdf.com>.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.
This page will not be added after purchasing Win2PDF.