

Assignment 3 - Software Quality Measurement

For this assignment, you will initially compare various software quality measurement techniques for more than one software development methodologies such as Extreme Programming, Unified Model, and Waterfall Model, etc. You will then determine when a software project is completed and ready to ship to customers based on the compared different quality measurement techniques. "How good is good enough?"

Background

You must write this report for the hypothetical software company *QualySoftTech Solutions*. The company is developing a software application for one of its clients. The software is designed to manage an electronic library that has lots of electronic documents, where the software should:

- Allow the client's user to input, remove and lookup entries.
- Provide an easy way to visualise the contents of the library.
- Documents to be "checked out" for modification and "checked in" when finished with.
- Allow for easy viewing of documents from a central repository.
- Allow client's users to copy items to their own drives. It will then keep track of what drives a document is kept on.
- Allow for consolidation (removing duplications) if the client's organisation is running low of space.
- Allow notes to be taken on any document. These notes can be restricted for viewing to certain users, or available to all users.

The Report

The report must be minimum 2000 words and includes (but is not limited to):

- ◆ Software quality attributes and definition of software quality
- ◆ Software quality planning and plans
- ◆ Software quality assurance
- ◆ Malcolm Baldrige National Quality Award approach to quality system
- ◆ ISO/9001:2000 approach to a software quality system
- ◆ SEI software development Capability Maturity Model Integrated (CMMI)
- ◆ ETVX (entrance, task, verification, exit) process model in software development
- ◆ Software productivity metrics and measurements
- ◆ Software quality metrics and measurements
- ◆ Software design and complexity metrics
- ◆ Software Project Assessments

You may have to make assumptions about the project for things that effect the choice. You should document these in your report.

Submission

Submit a single file called assign3.pdf/doc/rtf using the assignment submission page on WebCT.

No Implementation!

Note that you are not implementing any part of this project. You are simply evaluating software quality measurement techniques and methodologies.

External Readings

The text and the readings give you a very little insight into the above topic. You will need to do further reading on the topic, and it will extremely influence the outcome of your assignment. To get full marks on this assignment make insights into how each methodology will work *in this particular situation* and to extrapolate the characteristics of each to work out *what the consequences will be*. Any external material you do use must be referenced in your report.

Hints:

Development process for quality: comprehensive vs. agile process models

Structure in development models ranges from the simple “hack and build” to the complex, lock-step of the waterfall, cf., [Choose the Right Software Method for the Job](#). Currently, developers are divided as to how much structure is appropriate, and how it effects quality. Many suggest that the structure of RUP, CMMI, ISO 9000 is good, while other react to those “overly complex approaches” with simpler structures of Agile or eXtreme.

- Which process model produces better quality software, the comprehensive models, such as RUP or CMMI, or the smaller, agile models, such as Agile or eXtreme?
- Does it depend on project characteristics, such as size, type, domain, etc.?

Development model for quality: open-source vs. proprietary software quality

Proponents of open-source software have suggested that open-source software produces better software than commercial, proprietary development. In his essay on the open-source movement, "[The Cathedral and the Bazaar](#)," developer Eric Raymond wrote, "Given enough eyeballs, all bugs are shallow." (cf., [Building trust into open source CNET News.com](#)) Commercial software's quality problems are evidence. In response, Microsoft has redirected its efforts to improving quality (cf., [One year on, is Microsoft 'trustworthy'?](#), [The Trustworthy Computing Security Development Lifecycle](#)) Researchers have attempted to shed light on this controversy, e.g., [Is Open Source Software Development Faster, Better, and Cheaper than Software Engineering?](#), [High Quality and Open Source Software Practices](#). Why do you think?

- Which software development environment produces better quality software, the open-source model or the commercial, proprietary model?
- Does it depend on project characteristics, such as size, type, domain, etc.?

Quality assurance process: distinct and separate vs. integrated within development process

Some people suggest that software quality should be a distinct and separate process from the software development process (e.g., ch. 9 [Practical Insight into CMMI by Tim Kasse Artech House © 2004](#)). Yet, RUP does not include such a separate quality process. Few process models separate the quality management functions. There have been some attempts to “improve” RUP, e.g., [Why isn't there a RUP workflow for software quality assurance?](#) by Karen Ulferts.

- Which is the best approach, quality as an inherent part of development or as a distinguished, managing process?
- Does it depend on project characteristics, such as size, type, domain, etc.?

Quality improvement activity: requirements analysis vs. product testing

Read through any document on software quality, and you'll find that much of the text is devoted to code testing strategies. Yet, it is universally acknowledged that fixing requirements errors has a higher ROI than the downstream techniques. For example, if we could automatically derive programs from requirements (e.g., [Deriving Operational Software Specifications from System Goals](#)), and we had some assurance that the requirements were

correct (e.g., [Reasoning about Partial Goal Satisfaction for Requirements and Design Engineering](#)), then there would be no need to test the code, right? Why is there no software quality book dedicated entirely to software requirements? The lack of such books suggest that discovered problems are best fixed in the requirements, but that it is too difficult to discover requirements problems?

- How should the development resources (time, people) be allocated over the life-cycle (requirements, design, coding, testing) for the maximum benefit of detecting and removing software defects? Should more effort be focused upstream on requirements analysis or downstream, on code testing?
- Does it depend on project characteristics, such as size, type, domain, etc.?
- Does the software artifact type (e.g., structure, formalism) and available methods and tools play a role?