

SOFTWARE QUALITY THROUGH REQUIREMENT AND DESIGN

Massood Towhidnejad
Computer & Software Engineering Dept.
Embry-Riddle Aeronautical University
towhid@erau.edu

Abstract - One of the major areas of software engineering, as specified in the SWEBOK (SoftWare Engineering Body Of Knowledge) [1], is software quality. This paper presents an argument on why software quality should be an important component of an undergraduate computer science or software engineering curricula, and why it should not be viewed only as an exercise in software testing. Next we introduce a quality development life cycle with emphasis on attention to the quality during the early stages of the development life cycle. We look at quality during the design phase, and how different techniques such as inspection, and Fault Tree Analysis (FTA) can be introduced during the requirement and design phases in order to increase the quality of the software product.

Keywords: CS/SE education, software quality, inspection, software fault tree analysis

Introduction

Software is affecting our everyday life. Every time you use an electrical device, there is a very good possibility, that there is some sort of software behind the device mainly responsible for its control. For example, the TV you watch during the evening, the microwave you use to heat up your meal, the electrical razor that you used to shave this morning, or the car that you drove to get to home all have software for its control and operation. By the way when was the last time you catch a flight to go to vacation or a conference? Almost two million lines of code and over four thousand processors are used in the Boeing 777 aircraft. As you can see, our daily activities strongly depend on software, and soon we will have personalized robots that will be part of our households and will supply even more support and control of our day-to-day life. This could be a very scary picture, if we do not pay attention to the issue of software quality.

Now take a look at the software industry - with the exception of safety critical software, where people lives are at stake, industrial software does not have very high quality, (e.g., When was the last time your

application software “hanged up” and you lost everything you have done for the past couple of hours?). Of course when we talk about quality, we should make it clear, we are not just interested in whether software fails, but we also interested in the measure and assessment of software quality, and the method, techniques, and the processes needed to produce a quality product in an effective and efficient manner. Previously [2], we have talked about the quality of the product and process, and how one can incorporate these topics into an undergraduate software engineering and/or computer science curriculum. However, the purpose of this paper is to concentrate on the issues that affect the quality of the product. Traditionally, the quality of the product is only looked at during the testing phase of product development cycle. However, as it is learned by our industrial counter parts, the cost of detecting and removing a defect that is introduced during the earlier phases of the software development life cycle increase almost exponentially as we progress through the development life cycle. As it is shown in the following graph, based on the industrial data, the majority of the defect introduced during the software development life cycle is introduced during the requirement and design phase. Therefore, detection and removal of these defects during the same phase, that they are introduced, will significantly improve the quality of the product. There are two additional advantages to the early removal of the defects; these are reducing the time, and the cost of the software development.

Latest industry data (figure 1) indicates they are spending over fifty percent of their project budget on activities that would increase the quality of their software. Unfortunately, most companies spend these resources in the testing phase, chasing defects that have already been introduced in the front-end part of the development life cycle. This is a waste of time and resources, and industry leaders have shown that they can reduce testing and warranty costs by more than half, when they start implementing practices that forces

software quality control throughout the software development life cycle, with special emphasis in the earlier stages of development.

Defect Distribution

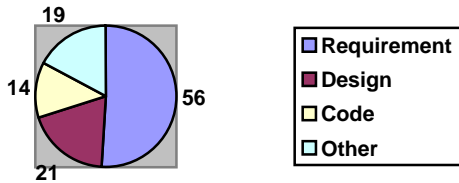


Figure 1. Defect Distribution

For example, a study produced by the Jet Propulsion Laboratory (JPL) [3] shows us that the cost of defect detection and correction increases ten fold as we move through each phase of the life cycle (requirements to design to implementation/test to release). Hence, the JPL study indicates the ratio of defect costs is 1:10:100:1000 (requirement: design: implementation/test: post release). This means a defect injected during the requirements phase would cost 1000 times more to fix during the post release than in the requirement phase.

As it was mentioned, the main purpose of this paper is to take a look at the issues that affect the quality of the product. This paper is mainly going to discuss a set of selected techniques such as inspection, and software fault tree analysis that could be introduced in an undergraduate computer science and/or software engineering program that could improve the quality of the software developed by the students.

Software Quality

Software quality is not a simple issue; its concepts and practices have been thoroughly studied and documented. For example, the SWEBOK has two chapters directly addressing software quality knowledge, Software Quality and Software Testing, which provide a detailed overview of quality concepts and practices, along with an extensive list of pertinent references.

The first step in determining the meaning of quality is to determine the necessary and desired quality attributes of the software product to be

developed. Generally, product quality attributes can be classified as functional, reliability, usability, efficiency, maintainability, or portability. Of course the most common quality attribute considered is the functionality. In order to achieve the quality goals for a product, a process must be set in place that defines, organizes and assesses the required quality assurance activities. Quality activities cover a whole spectrum of verification and validation (V&V) techniques, which include inspections, reviews, tests, and audits of software artifacts produced in the development process.

A Quality Model

Figure 2 outlines the principal elements of the so-called “V-Model for Quality.” The V-Model for Quality is a modification of the original V-Model software development life cycle originally introduced in 1997 [4]. The rectangular elements in the figure represent the traditional phases in software development (requirements and design on the left and testing on the right). This model contains features that

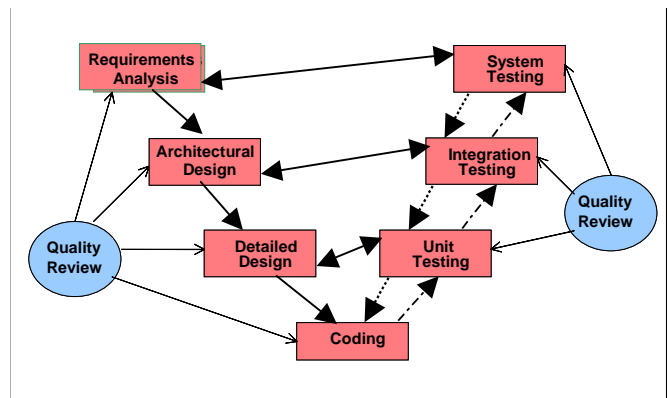


FIGURE 2: V-Model for Quality

have significant implications and ideas for designing curricula, which will help students, learn how to produce high-quality software products in an effective manner. First and most obvious is that the model advocates that one carry out a problem analysis and high-level design before proceeding to implementation and testing. Unfortunately, there are so many examples; both in industry and academia, of the emphasis on a code-and-test mentality, that we need to state the obvious: concentration on the front-end part of development can produce a higher-quality product at a lower cost [5]. Of course this does not necessary implies that no code is written, until all the requirements are collected and verified, the detail

design for the whole system is completed, and verified, in another word, we should only follow the traditional waterfall software development life cycle to its fullest. In contrast, this model supports any software development life cycle, such as spiral, agile, etc. The slanted double-headed arrows between the rectangles indicate parallel activities. For example, the two rectangle in the top with the corresponding double-headed arrow indicates, that once the requirement analysis phase started, then almost at the same time, the planning phase for the system testing starts, and these two activities complement each other in order to make a better requirement document and system test plan. As it is shown, the testing activity is divided to two distinct paths, the planning path, which is identified by dashed arrows from top to bottom, and the execution path, which is identified by the dashed arrows from the bottom to the top. Highlighting these paths enforce the idea of start the testing from the beginning of the development, and not just after the completion of coding. Therefore, the curriculum implications of this part of the model are as follows: students should learn about and do system test planning at the same time as they learn how to analyze and specify software requirements; teaching integration planning should correspond with the teaching of high-level design; and unit test planning should be taught along with detailed design of a unit.

Finally, the “quality review” bubble in the model emphasize that the deliverable(s) produced in each development activity (software requirements specification, software design specification, code, system test plan, integration plan, unit test plan) should be subject to a quality review.

The activities that are discussed in the remainder of this paper will best fit in the Quality Review bubbles. These activities are very important in increasing the quality of the product, but unfortunately, since they are not directly tied to the development of the product (i.e., no specific artifact is generated that the customer will pay for it), they are usually ignored or the effort spent in these activities are reduced in the industry. However, we as educator should have the responsibility to inform our students about the benefits of such activities, so when they graduate and join the work force in the industry, they can become the change agent, and increase the awareness of their colleagues to the importance of them.

In the following section, we first briefly introduce the concept of review and inspection, followed by the

software fault tree analysis. These discussions are then followed by how these techniques can be introduced during the design phase to improve the quality of the software.

Inspection

Unfortunately, inspections, reviews, and walkthroughs are sometimes used interchangeably, and they all refer to a set of activities that are performed at the completion of a development phase (i.e. requirement, design, code, etc). They all behave as a filter to detect defects early on the development life cycle. However, inspection is very formal process with well define activities, where the reviews and walkthroughs are less formal. Due to its formal process, inspection is highly effective in detecting defects in the product, however its effectiveness drastically decreases as the process is modified (streamlined). Formal inspections are very costly, for this reason, it is not uncommon for an industry to not include formal inspection as part of their development life cycle. Therefore, it is critical to teach our students what is a formal inspection, how it can be done, and what are the main advantages of it. Figure 3 represents a simple view of formal inspection process. There are seven steps in a formal inspection, these are, preparation, overview, individual preparation, formal meeting, rework, follow up, and inspection (causal) analysis.

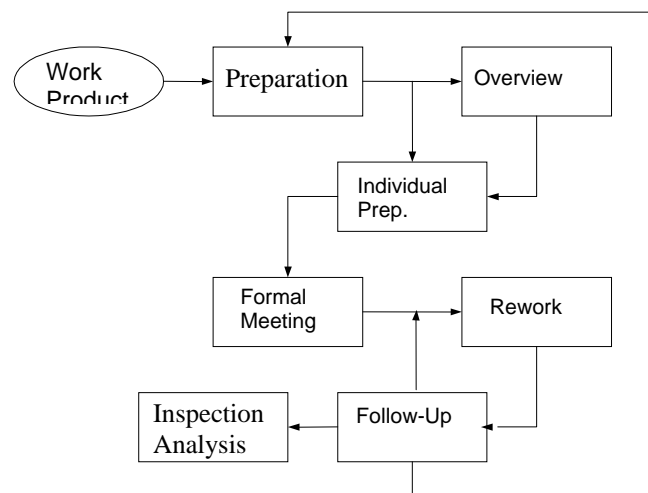


Figure 3. Inspection Process

During the preparation stage, the moderator (person responsible for the inspection) perform a preliminary inspection of the product to make sure the product is ready for the inspection (all material are

available, and of relatively good quality). In addition, the moderator identifies all the participants in the inspection. It is critical to include people who are not part of the development team as inspector, and whenever possible include a customer representative. The moderator distributes the artifact under the inspection to all the participants and it also provide them with a checklist which should be used for the inspection and the type of defects that are being sought. Finally, the moderator establishes a schedule for inspection.

The purpose of overview is to introduce the inspector to the product under the inspection. During the overview meeting, the author of the document under the inspection will provide the necessary background on the artifacts, its functionality, the methods, techniques, and procedures that were used in producing the artifact. This is an opportunity for the inspectors to ask for clarification.

The individual preparation is the most critical step of the inspection process. This is where the individual inspector reviews the document for defects, and produces their major and minor defect list.

During the formal meeting, the moderator first check with the inspectors and collect their defect list, with their corresponding time they spent inspecting the artifact during their individual preparation. Then the reader will go over the document (depending on the work product this may be done page by page, or section by section or any other combination), and each individual identify the major defects that they found in the product. The recorder collects this data, and at the completion of the formal meeting submits this information to the moderator. The moderator communicates both the major and minor defects found by the inspection to the author.

The author will attempt to fix the defects identified by the inspector during the rework stage, each defect that is fixed as marked as such. If the author does not agree with a defect that is identified, then he should mark that defect as not fixed and communicate the reason why the defect is not fixed with the moderator during the follow-up stage.

Software Fault Tree Analysis

Fault Tree Analysis (FTA) [6] is a technique used in the area of reliability. Initially, FTA was introduced in the 1960s, with the primary purpose of identifying those circumstances that could cause a system to reach

a hazardous or unsafe state. FTA is powerful static analysis tool. Given a specific hazardous state, FTA uses a *backward* (also referred as *top-down* or *deductive*) search technique in order to identify conditions that would cause the system to reach that state. In other words, once a specific hazard is identified (hypothesized), FTA will search all possible combinations of the conditions (initial states) that could force the system to reach that state. FTA is typically applied to hardware systems, but recently attempts have been made to apply FTA to software artifacts [7,8,9].

Software Fault Tree Analysis (SFTA) could be applied during the requirements and design phase to identify the critical component of the software where safety and hazardous states are the major concerns. Once these critical components are identified, special attention is given to the flagged components during the development and verification and validation activities.

SFTA at requirements phase

The main objectives of applying SFTA during this phase of software development are to:

- Identify weaknesses that exist in the requirement specification. Weak requirements will either be modified or additional requirements will be added in order to eliminate or mitigate these weaknesses.
- Identify all the requirements that have a direct effect on the safety of the system. This can be done either through the knowledge collected as part of the requirements elicitation, or identifying the pattern of use and the surrounding environment that could affect the software, by forcing it to a hazardous state. Once requirements with safety considerations are identified, these requirements will be traced throughout the development life cycle. It is assumed that a requirement traceability matrix is included in the software development artifacts to help with this task.

SFTA at design phase

The main objectives of applying SFTA during this phase are to:

- Identify the weaknesses of the high-level design. At this stage, appropriate modifications will be implemented in order to strengthen the overall design.

- Identify the components/modules and subcomponents that have direct effect on software safety. These modules and those implementing the requirements with the safety consequences are identified. Then, special attention may be given to the generation of their implementation, by guaranteeing the elimination of design factors that could force the system into a hazardous state.

Applying SFTA during the detailed design phase will produce the best return on investment. It is here that a software product exists in its most ideal form for SFTA to be applied. Software is represented in the form of some number of modules where functionality, interfaces, inputs, and outputs are well defined. This is the closest we get to representing software structure in a way that is analogous to hardware modeling, a point prior to development where the salient system features, i.e., gates, encoder, functionality, interfaces, inputs and outputs are well defined. The same can be said about a software system at the detailed design phase. Here the software is represented with an equivalent amount of detail that we can achieve the equivalent degree of insight. Applying SFTA at this point enables us to identify modules (objects, methods, or functions) that could directly affect the safety of the system.

In both the preliminary and detailed design phases, once a module or a set of modules is identified as having possible impacts on the safety of the system, additional safeguards need to be embedded into the design in order to guarantee their safe operation.

SUMMARY

As an educator, we have the responsibility to educate our students to best of our abilities so they can become a positive contributor to our society. As software becomes an integral part of our life, we have to make sure we produce computer scientists and software engineers who are conscious about the quality of the products that they produce, and provide them the knowledge and expertise that are necessary to produce high quality products. This paper, first introduces the V-Model for quality software development life cycle, and provides a brief discussion of two techniques, inspections and software fault tree analysis that could easily be incorporated in the undergraduate education. Introduction of these concepts throughout the undergraduate curriculum increases the awareness of students to the importance of software quality.

REFERENCES

- [1] Bourque, P. and Dupuis, R., eds., *Guide to the Software Engineering Body of Knowledge*, Trial Version 0.95, <http://www.swebok.org/>, May, 2001.
- [2] Hilburn, T., Towhidnejad, M., "Software Quality Across Curriculum", Proceedings of 32nd ASEE/IEEE Frontier in Education Conference", November 2002, pp .
- [3] Rothman, Johanna, "What Does It cost to Fix a Defect?", Column Archive, StickyMinds.Com, <http://www.stickyminds.com/>.
- [4] *V-Model 97, Lifecycle Process Model -Developing Standard for IT Systems of the Federal Republic of Germany*, General Directive No., 250, June 1997, <http://www.scope.gmd.de/vmodel/en/>.
- [5] Humphrey, W. S., *Introduction to the Personal Software Process*, Addison-Wesley, 1997
- [6] NUREG-0492, Fault Tree Handbook, U.S. Nuclear Regulatory Commission, January, 1981.
- [7] Lutz, R. R., "Targeting Safety-Related Errors During Software Requirements Analysis," *Journal of Systems and Software*, 1996, 34, 223-230.
- [8] Helmer, G., Slagell, M., Honavar, V., Miller, L. and Lutz, R., "A Software Fault Tree Approach to Requirement Analysis of an Intrusion Detection System" *Symposium on Requirements Engineering for Information Security*, March 5-6, 2001.
- [9] Towhidnejad, M., Wallace, D., Gallo, A. "'Fault Tree Analysis for Software Design", 27th Annual IEEE/NASA Software Engineering Workshop, December 2002.