A&QT-R 2002 (THETA 13)
2002 IEEE-TTTC International Conference on Automation, Quality and Testing, Robotics
May 23 – 25, 2002, Cluj-Napoca, Romania

SOFTWARE QUALITY

Silviu FRANȚESCU, Emil VOIȘAN, Ovidiu DOBREANU, Cosmin CORNEA, Gabriel FAUR, Tudor PERENI

Department of Automation and Industrial Information
Faculty of Automatics and Computer Sciences
"Politehnica" University of Timisoara
Bd. V. Parvan, no. 2, 1900, Timisoara, Romania
Tel.: 40-56-204333
Fax: 40-56-192-049
Email: sfrantescu@computervoice.ro, evoisan@aut.utt.ro, odobrean@aut.utt.ro, ccornea@aut.utt.ro, fg1454@aut.utt.ro, tudor.pereni@alcatel.fr

Abstract: In the recent years software development has become more and more of a bottleneck due the pressure to decrease product development time while increasing the quality and functionality for the software is expected. In this article we'll take a look at why the issue of software quality should be the primary thought whenever you're programming, as well as discussing some methods for building high-quality software systems.

Keywords: software, quality, errors.

1. INTRODUCTION

The software problems especially their quality aspects were discussed before [1], [2] and will be more. This article is just trying to remind the software developers the main problems that can appear in a software development process and the possible causes.

Software quality can be defined as the conformance to explicitly stated, functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software. However, quality is a subjective term because it depends on who the customer is. A wide-angle view of the possible customers of a software development project might include end-users, stockholders, future software maintenance engineers, government departments, etc. Each type of customer will have their own opinion about quality – the end-user might define quality as user-friendly and bug-free, while an accountant might define quality in terms of profits.

Most programmers are superficial about controlling the quality of the software they write. They write some code, run it through an ad hoc test, and if it seems ok, they are done. Or, they use some code developed by someone else, and assume the code is in good condition and can be used, without further modifications and tests. What are the results of such activity? On June 4 1996 the first flight of the European Space Agency's new Ariane 5 rocket failed shortly (in 4 sec.) after launching, resulting in an estimated uninsured loss of a half billion dollars. It was reportedly due to the lack of exception handling of a floating-point error in a conversion from a 64-bit integer to a 16-bit signed integer. And there are dozens of such examples.

Some organizations have coding 'standards' that all developers are supposed to adhere to, but everyone has different ideas and a general accepted standardization wasn't achieved yet. There are also various theories and metrics, such as McCabe Complexity metrics. Also an excessive use of standards and rules can decrease productivity and creativity.

2. SOFTWARE QUALITY ASSURANCE (SQA)

SQA involves the entire development process. It monitors and improves the process, verifying that the set of agreed-upon standards and procedures is followed, and ensuring that the problems are found and dealt with. With other words, it is oriented on prevention. The overall goal of SQA is to minimize the cost of fixing problems by detecting the errors early in the development cycle.

Let's look at some stages of structured software development. No matter what software development cycle we will follow, we always have to interact with requirement analysis, system specifications, system design, code implementation, testing and maintenance. So these stages are somehow universally applicable. For small projects some of those stages are not strongly required, however the designer will benefit by knowing and using them and educating himself to the most important aspects of the development stages.

2.1 *Requirements Analysis*

- The system requested by the customer should be feasible.
- The requirements specified by the customer should satisfy his real needs, thus recognizing requirements that are mutually incompatible, inconsistent, ambiguous, or unnecessary.
- Give the customer an idea about how the software system will be build in order to address the requirements.
- Avoid misunderstandings between the developers and the customer, which can lead to further problems.

2.2 *Specifications*

- The specifications should be consistent with the requirements.
- The specifications should cover system's flexibility, maintainability and performance.
- A testing strategy should be established.
- A realistic development schedule should be established, including periodical reviews
- A formal change procedure should be created for the system. Uncontrolled changes, resulting in many versions of the system, will contribute to quality degradation.

2.3 *Design*

- The changes made to the system's design should be properly controlled and documented.
- The coding process should not start until de system design components are approved according to requirements.
- The design reviews should proceed as scheduled.

2.4 *Coding*

- The code should follow the established standards of style, structure, and documentation. Even though some programming languages (C/C++) lets us be very compact, and super-obscure in coding, we should bear in mind that we or others may have to work with that code again some day. Clarity of code is preferred over conciseness.
- The code should be properly tested and integrated, and the revisions made are properly identified.
- The code writing should follow the schedule. The customer should be informed of the delays, and to know about the impact on the delivery time.
- The code reviews should be made on schedule.

2.5 *Testing*

- Testing plans should be created and followed. This includes a library of test data, driver programs to run through the test data, and documentation of the results of each formal test that has been performed.
- The test plans created should address all of the system specifications.
- After testing and reworking, the software should "perform" according to the specifications.

2.6 *Maintenance*

- The code and the documentation should be consistent. We tend not to update the documentation when we change the program. Such an oversight can create nightmares the next time someone will have to make a change.
- The changes made in the code should follow the coding standard, and do not deteriorate the overall code structure.
- A configuration control should be observed, in order to integrate the changes into the production version.

At this point we can formulate a few important ideas. First a product has to be built considering the quality as the primary objective. There is no point in testing a product to check if it fits the quality standards or not, if that product was not designed/built following some quality standards. Second, the quality related issues should be managed by persons outside the project development core. Both, the developers and the customers should be concerned about the quality assurance of the product, in this way both being involved in the development process, to a certain degree, and sharing also some responsibilities in order to obtain a product according to the specifications and following the agreed-upon standards.

3. SOFTWARE QUALITY AND THE HUMAN FACTOR

There are two ways to obtain software free of errors. The first one is to prevent the introduction of errors. The second is to identify the bugs, which are more or less hidden in the code, and eliminate them. Obviously, the first method is superior.

If the specifications are lousy, and consequently, the design poor, the programmer will have a difficult job understanding and implementing what the customer wants. The errors are more likely to appear in the coding part of the development process.

A particular classification of coding errors is to classify them into syntax errors and logical errors. The syntax errors are as easy to catch, as common they are. They depend on the programming language used, however on the industry there are two main types of programming languages used: assembly-level programming and high-level language programming.

Assembly-level programming is preferred for real-time systems, due to a faster execution and less memory requirements. On the other hand, assembly-level languages are hard to maintain. There are a huge number of instructions required to perform a given operation for a specific processor.

Considering the high-level languages, they are far easier to work with. A high-level programming language comes closer to the human language. They translate the written high-level code into an assembly-level sequence of code, using a compiler. The drawback is that the software created by a high-level language runs slower and requires more computing resources. One thing worth mentioning is that the compiler has error detection capability. This means that logic, syntax and other coding errors can be detected and corrected before an attempt is made to load and run the program.

Such error detection capability implemented into the compilers was necessary in order to compensate the human failures. For a given operation these failures depend on a large number of factors. This factors can be grouped under three main categories: intrinsic, environmental and stress.

These categories will be detailed as follows.

3.1. Intrinsic Factors

These factors covers the basic characteristics of individuals, in our case all persons involved in the development of a software project:

- Motivation: such person should have a proper motivation for a specific job.
- Physical ability: such person should be physically capable to perform the operation.
- Mental ability: a person should have the basic intelligence required to perform the operation.
- Temperament: a person should remain calm under stress.
- Concentration: a person should be capable to focus on the job.
- Speed of response: a person should be quick enough to respond in an emergency situation.
- Knowledge: a person should possess sufficient knowledge in order to carry out the operation correctly.

The persons directly involved in the development process should be selected based on the characteristics required by the project. The selected persons should attend a course of training that will give them the technical knowledge and experience necessary to provide good quality software.

3.2. Environmental factors

The working environment has a big influence on the persons involved in the project development, in particular the software developers. The environmental influences can be physical, organizational, and personal. Physical factors include temperature, humidity, and noise level. Organizational factors include relationships with colleagues, relationships with the project managers, job satisfaction, salary, job security, promotion prospects, vacations. Personal factors as hunger, thirst, tiredness, physical and mental health, home life. Such factors are in tight relation with the organizational factors.

3.3. Stress factors

The quality of software produced by developers depends on their stress level. Figure 1 shows the relation between human error rate and stress level. It can be seen that there is an optimum stress level at which the error rate is minim. If the person is either bored or overexcited, the error rate increases.

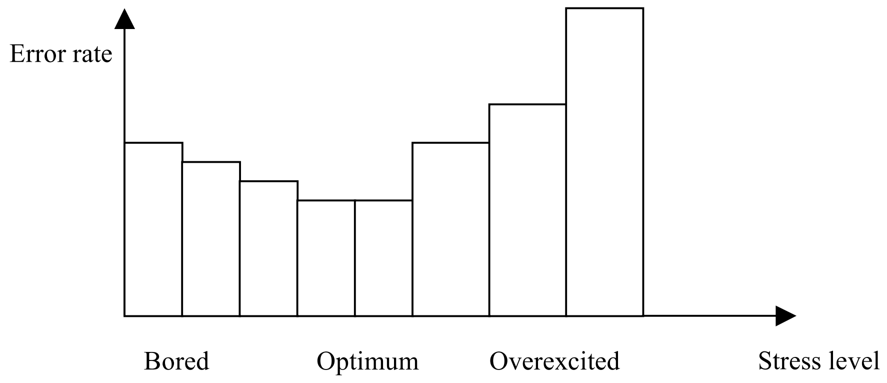

Figure 1. Error variation with the stress level

Error rate increases significantly due to several factors: the development's schedule is not followed, the customer too often changes project requirements, an unrealistic evaluation of project's requirements.

In 1988 a Russian satellite was lost on its way to Mars. Why? According to Science magazine, "not long after the launch, a ground controller omitted a single letter in a series of digital commands sent to the spacecraft. And by malignant bad luck, that omission caused the code to be mistranslated in such a way as to trigger the test sequence". And Phobos was never recovered. Human error it is a fact of life. People are not precision machinery designed for accuracy. The natural tendency to interpret partial information can cause operators to misinterpret system behavior in such a plausible way that the misinterpretation can be difficult to discover.

4. CONCLUSIONS

Current-generation software of medium to high complexity has to meet quality requirements along with other specific characteristics. Those characteristics include, for example, safety, reliability, reusability, predictability, portability, and the like. Characteristics are an important (if not essential) component of user requirements. Oddly enough, however, characteristics are often insufficiently defined, implemented, or verified, thereby presenting a serious threat to project success.

Because of the large use and reliance on software systems today, there is a great need for effective quality assurance alternatives and related techniques. According to the different sources of errors, they can be classified into two categories: error prevention and error reduction. Existing software quality literature generally covers error reduction techniques such as testing and inspection in more details than error prevention activities, so before anything the most important course in a project development is to prevent errors not to reduce them. In order to acquire this objective it is required to follow a model in software development: in early stages in order to obtain the specifications and prototypes correctly it could be used the rapid prototyping, once the specifications are known the development can proceed using the waterfall model (Figure 2):

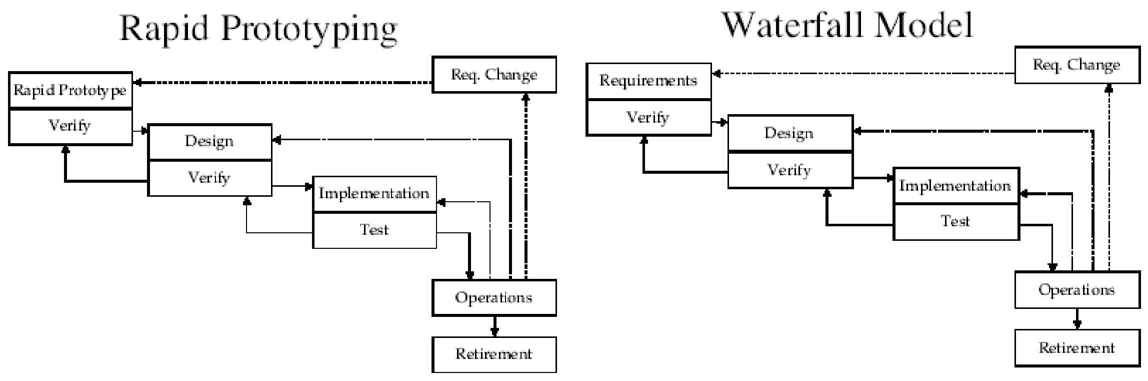

Figure 2. Software development models

5. BIBLIOGRAPHY

1. Bentley, John, [1999] Introduction to Reliability and Quality Engineering, Addison-Wesley Longman.
2. Pressman, Roger S., [1987] Software Engineering: A Practitioner's Approach, 2nd Edition, McGraw-Hill.
3. Anderson, Kenneth M. [2001] Foundations of Software Engineering.
4. Stoicu-Tivadar, Vasile Software Engineering for Process Control.
5. Waldrop, [1989] M. M. Phobos at Mars: A dramatic view -- and then failure. *Science*