Seminar paper

Software Quality

Testing: Concepts, Issues, and Techniques

Micha Waterböhr

June 4, 2013

Lecturer: Prof. Dr. Gerd Beuster

Index

1 Abstract

This seminar paper will show how testing is integrated in the overall process of software quality engineering. Since quality assurance plays a major part in this process the basic ideas of quality assurance will be explained. One of the most common tools for quality assurance is testing. Therefore an introduction to the topic of Software Testing will be given. The main content of this paper will then describe methods of testing which can be used to classify different testing techniques and put them into perspective. Questions about what to test and when to stop testing will be discussed.

At the end we will give a short overview and description of different types of testing which are most commonly used to ensure a certain level of quality assurance. The goal is to give a comprehension of the importance of software testing and some basic knowledge of the concepts and issues of testing.

2 Introduction

In our modern society software systems are used everywhere around us. In our daily life we rely on the proper functioning of software systems. The economy, our private life, work, transportation, even some social functions are ruled by the use of software systems. These systems are becoming more and more complex.

But how do we make sure that those systems work according to the user's expectation? Let us just think of some examples where the correct functioning of software is indispensable. The first example that might come to mind is health care where software systems are used for medical purposes. Just imagine what would happen if the improper functioning of some medical equipment would cause some damage or even death of a person.

The malfunctioning of software installed in an airplane could cause a disastrous outcome. But even on a lower level the outcome might be discontent customers leading to the loss of good reputation for a certain business. This could even result in the breakdown of the said business.

This makes it necessary to find ways to make sure that those defects might be prevented, reduced or contained. Testing is one of the greatest tools to ensure software quality. Testing in fact is essential in order to produce software with a high level of quality and reliability.

3 Quality Assurance

Talking about software quality always leads to the topic of quality assurance. How can we be sure that the software system has quality we want it to have?

The major activities of quality assurance are centred on ensuring that only few if any defects remain in the software system when it is delivered to the customer. But since it is not always possible to eliminate every single defect, thus having flawless software, quality assurance also includes dealing with those still defects still remaining in the system. The goal is to contain those failures or to reduce the resulting damage. This is known as defect containment. Even before coding the software defect prevention activities are performed to reduce the chance of injecting faults into the software. However, the main task will always be the reduction of defects.

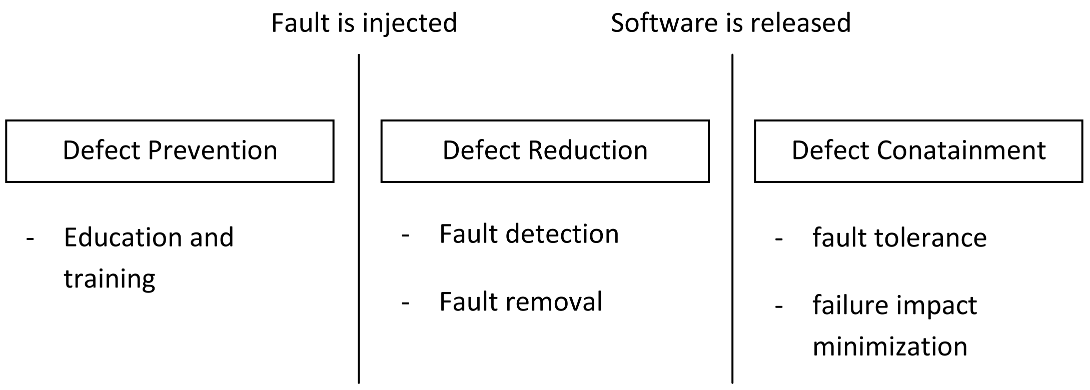

Figure 1 - Quality Assurance activities

3.1 Defect Prevention

The goal of defect prevention activities is to reduce the chance that faults or defects are injected into the software in the first place. The target of defect prevention activities is to eliminate the error source. The first step would be to determine the error source that leads to fault injections. Some of these error sources might be human misconceptions, incorrect designs or deviations from the product specification, or the disregard for selected standards.

In case of human misconceptions a way to prevent defects is to train and educate the people involved in developing the software. This will prevent certain types of faults from being injected into software products. Tian broke this down into specific areas of knowledge that need to be conveyed to the developers [TIAN05, p. 32]:

1. Product and domain specific knowledge
2. Software development knowledge and expertise
3. Knowledge about Development methodology, technology and tools
4. Development process knowledge

To prevent the software from deviating from product specifications formal methods can be used to ensure, that the product specifications are met. Formal methods include formal specification and formal verification. A very influential formal method known as the axiomatic approach uses so called pre-conditions and post-conditions to describe the program state before and after the execution thus allowing us to see if the software is working according to these specifications. The problem with these formal methods is the high cost which is involved due to the hard task to develop these formal specifications correctly without automated help and it is often difficult to prove the formal verification.

3.2 Defect Reduction

As shown in figure 1, the main tasks of defect reduction are to detect faults that are already injected into the software and then to remove the faults that are found. Fault detection is the main part of defect reduction. Once the fault is identified it is easier to remove the fault since the location where the problem is embedded is known. Testing is the main instrument to find these faults. There are different testing techniques for different types of faults. We will discuss the process of testing in general later on in more detail.

Besides testing inspection is another way to find faults. In inspection, software is examined through a human inspector who critically reads and analyses the software code. Typically there is not only one inspector but many. This technique leads to immediate fault removal.

3.3 Defect Containment

Defect containment takes effect after the software is released and targets those fault that remained in the software and are causing failures while running or executing the software.

Defect containment minimizes the failure impact and thus contains it in a sense. This is very important especially for safety critical systems.

Using software fault tolerance is another way to deal with faults that still remain in the software system. A general method to achieve fault tolerance is to use redundancy, or spare parts, in the program so that the system might stay operational even in the case of failure.

4 Testing

In a very simple way testing can be viewed as executing a program or software and look at its behaviour or outcome. This obviously requires executable code. But this is more a form of informal testing, which is done for example when the customer is executing the software himself.

Testing also requires the human tester of course and also the time it takes to test. This will obviously cost money and so software testing will always be a trade-off between budget, time and quality.

Motivation

As discussed in the previous chapter quality assurance is the main purpose of software testing. But originally testing was primarily used to demonstrate functionality or in other words to show that the software works correctly [TIAN05, p. 68].

Another aspect of testing is to ensure that software is able to compete economically. If software is full of faults is it most likely that this software is not able to compete with other software. Also this might damage the reputation of the developing business.

A totally different angle is shown if we look at the possibility of saving time and work as testing is a tool to find faults early in the developing process where it is a lot easier to correct the faults in contrast to correcting them later on when they are discovered by customers.

The ISO standard [ISO/IEC 25010] gives a quality model that is comprised by the following attributes that a piece of software should fulfil in order to reach a certain level of software quality: functionality, reliability, usability, efficiency, maintainability and portability, security and compatibility.

- Functionality is achieved through the existence of a set of functions and their specified properties.
- Reliability is achieved though the capability of software to maintain its level of performance under stated conditions for a stated period of time.
- Usability represents a set of attributes that bear on the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users.
- Efficiency is the relationship between the level of performance of the software and the amount of resources used, under stated conditions.
- Maintainability is measured with the effort needed to make specified modifications.
- Portability is the ability of software to be transferred from one environment to another.

- Security includes characteristics like confidentiality, integrity, nonrepudiation, accountability and authenticity.

- Compatibility includes the characteristics co-existence and interoperability.

Testing is an excellent way to ensure that software acquires these attributes.

4.1 Testing Process

Even though all major test activities are centred on test execution there are two other groups of activities that we will consider being part of the testing process.

The following figure shows the chronological order for testing activities:

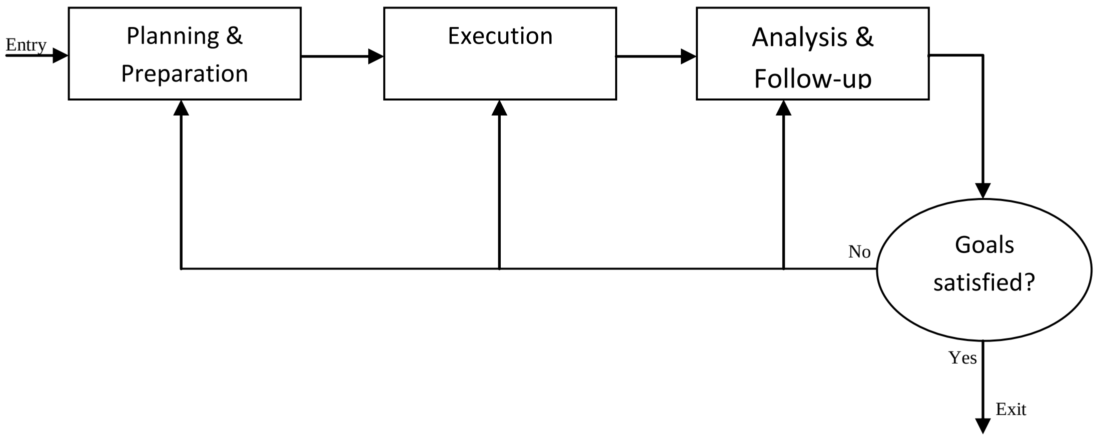

Figure 2 – Testing Process (compare [TIAN05, p. 69])

Because of the increasing complexity of software planning and preparation activities are inevitable in order to be able to meet certain standards of software quality. Activities that are included in planning and preparation are:

- Information gathering

- Goal setting

- Model construction

- Test case preparation

- Test procedure preparation

It is very important to set specific reliability or coverage goals. These goals will then be used as the exit criterion for the testing activities. The test case preparation and test procedure preparation will then be used as input for the execution activities. Preparing test cases is naturally associated with test preparation. There are different methods of generating test cases automatically and it is often necessary to select test cases based on some formal models since it is not always possible to execute a program with every single test case.

In order to effectively execute many “test-runs” it is necessary to make sure that one failed test-run will not block the execution of the other test runs.

After execution, the analysis and follow-up activities begin. The results from the test that is being executed will be checked and analysed to see if the outcome is as expected or if some failure has occurred. Knowing that an error did occur though is not the same as knowing exactly where it did occur. So it is important to use every single detail of information from the test results in order to be able to identify the location of the fault. Once this is accomplished the fault can be removed.

4.2 When to test

Putting the process of testing into the bigger picture we will look at a figure depicting the waterfall model which is commonly used. Even with a different model, other than the waterfall model, the general idea stays the same.

As it is shown in figure 3 the testing phase follows right after the coding phase. But the focus of defect reduction or defect removal already overlaps with the coding phase.

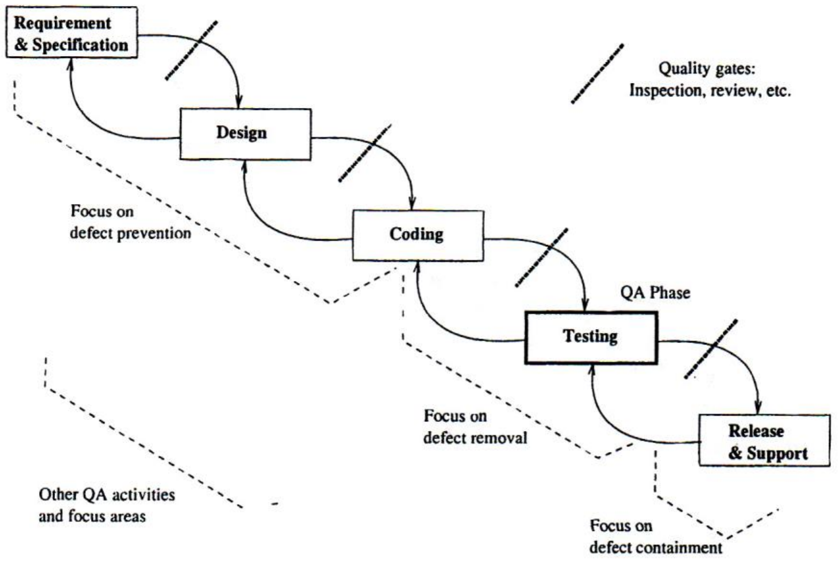
Requirement & Specification
Design
Coding
Testing
Release & Support
Quality gates: Inspection, review, etc.
QA Phase
Focus on defect prevention
Focus on defect removal
Focus on defect containment
Other QA activities and focus areas

Figure 3 – QA activities in the waterfall process (source [TIAN05])

As explained in chapter 3 testing activities are split into *planning & preparation*, *execution* and *analysis & follow-up*. Planning and preparation activities can be performed much earlier. Even as early as in the design phase you could start planning for testing.

An example for such testing is Extreme Programming.

One of the fundamentals of Extreme Programming is "w*riting unit tests before programming and keeping all of the tests running at all times."* [BECK99]

"Erich Gamma coined the phrase "Test Infected" to describe people who won't code if they don't already have a test. The tests tell you when you are done—when the tests run, you are done coding for the moment. When you can't think of any tests to write that might break, you are completely done." [BECK99]

This just shows that testing, since we defined test preparations as part of testing, indeed can begin long before code is created. But in order to see the possibility that testing is possible after product release we just need to think about informal testing by the customer as he is executing the program over and over and maybe finding some still existing fault which then can be addressed through the support for the product. As we can see testing is not confined to the testing phase only. Anyhow, most parts of testing activities will always happen in the designated testing phase.

4.3 What to test

As to know what to test we will differentiate between two methods of testing: Black-box testing and white-box testing. Later on we will see that most testing types or techniques can be categorised into one of these two methods.

4.3.1 Black-box testing

"Black-box (or functional) testing verifies the correct handling of the external functions provided or supported by the software, or whether the observed behaviour conforms to user expectations or product specifications." [TIAN05]

Black-box testing is a software testing method in which the internal structure, design, and implementation is not known to the tester. This method is named so because the software program, in the eye of the tester, is like a black box. One cannot see what is inside.

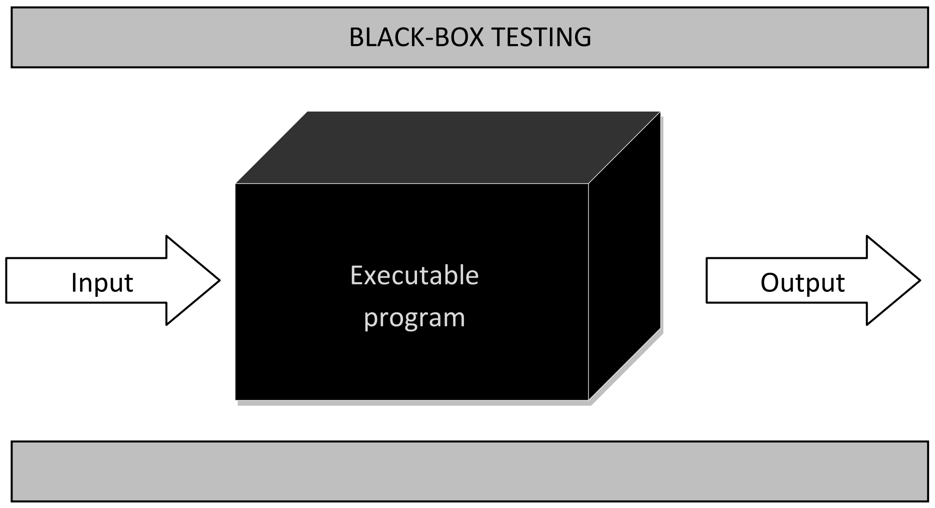

Figure 4 – Black-box testing

Even just running the program and making observation of its outcome is a form of black-box testing. This form is also known as “ad hoc” testing. An example of “ad hoc” testing is if the customer is running the software and if a failure occurs he will hopefully report the problem and it can be analysed and fixed.

The emphasis of black-box testing lies on reducing the chances that customers encounter functional problems. The more black-box test is applied the lesser the chance that a customer will randomly run into problems or failures since they could have been removed through the testing already. The testing process mentioned earlier can be applied to the black-box testing method. Then the focus in planning is to identify all the external functions that need to be tested.

Black-box testing is typically used to test large software systems which then act as the black box. Since the whole program or unit which is to be tested needs to be executable, this testing method is also used in rather late sub-phases of testing. This testing method is very effective in detecting and fixing problems of interfaces and interactions. Since the tester does not need to know the implementation details this method is also used by third party personnel or professional testers.

4.3.2 White-box testing

“White-box (or structural) testing verifies the correct implementation of internal units, structures, and relations among them.” [TIAN05]

With the white-box testing method the internal structure, design, and implementation is known to the tester. In contrast to the model of the black-box we talk about a “white-box” even though it would be more accurate to say “transparent-box” or even “glass-box” since the inside of the box is visible.

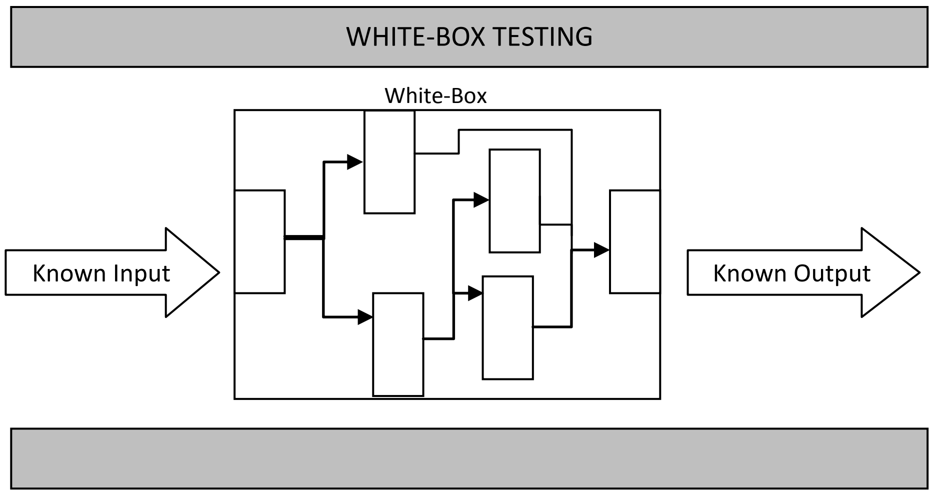

Figure 5 – White-box testing

Probably the simplest form of white-box testing is *coverage statement testing* by using some debugging tools to go through the program step by step and check the state of the program after each statement. Here the tester can see exactly which statement is being executed and can see if the outcome is according to the expected outcome. If this is not the case the fault has immediately been found and can be fixed right away, since the location is also known. Only what is present in the code can be tested and accordingly it is hard to detect problems of omission or design.

In order to effectively use this method it is required that the tester has a good understanding of the code or is familiar with it. For this reason the person testing is most likely the programmer himself. This dual role makes defect fixing much easier.

The white-box testing method is used for rather small objects und can consequently be used in early sub-phases of testing.

4.4 When to stop testing

Probably the most important question about software testing is when to stop testing. A resource based approach would be to stop testing when you run out of time or when you run out of money. But this would be irresponsible if the goal is to achieve software quality.

Without defining some criterion we could test indefinitely and would never come to a realistic stop. Normally testing stops when product is released and determines what level of software quality the customer might expect. The decision about the exit from testing is associated with achieving quality goals, in our case reliability or coverage goals. Software should only be released if those goals have been achieved.

Two useful testing techniques that work to achieve these goals are *usage based testing* and *coverage based testing*. The difference between these two techniques is the stopping criterion used and the perspective from which the object is seen.

4.4.1 Usage based statistical testing

"Usage based statistical testing views the object from a user's perspective and focuses on the usage scenarios, sequences, patterns, and associated frequencies or probabilities." [TIAN05]

With the usage based testing we look at a number of uses of the software. We speak of "statistical" usage based testing because statistical sampling is needed due to the massive number of different usage patterns. The overall testing environment resembles the actual operational environment and the execution of specific test cases in a test suite resembles the usage scenarios, sequences, or patterns with which a customer could possibly use the software.

The usage information is captured in so called “operational profiles”, short OPs. The operational profile is a quantitative characterization of how the software will be used. There are two different approaches to OPs.

The first approach is the model of flat OPs, which was first described by John Musa. According to Musa a profile is a set of independent possibilities called elements, and their associated probability of occurrence. It is a practical approach to ensure that a system is delivered with a maximized reliability, because the operations most used also have been tested the most. [MUS93]

The second approach is the Markov chain[1] based usage model, or short Markov OPs. In this approach finite state Markov chains are used to model the sequences. The states of the Markov chain represent inputs to the software system, while the arcs imply an ordering of the inputs and are annotated with probabilities.

Utilising these models ensures a certain level of reliability and as soon as one reaches the reliability goal which is used as the stopping criterion one may stop testing.

4.4.2 Coverage based testing

“Coverage based testing views the objects from a developer’s perspective and focuses covering functional or implementation units and related entities.” [TIAN05]

The stopping criterion for coverage based testing is some form of test coverage. The simplest form could be a checklist of all the major functions of a program. Once all functions on the checklist have been tested the desired coverage has been fulfilled and one my stop testing.

For most systematic testing techniques simple checklist are not enough. In such cases some formal models are used.

Tian defines the generic steps and major sub-activities for coverage based testing as follows [TIAN05]:

- Defining the model
- Checking individual model elements
- Defining coverage criteria
- Derive test cases

Coverage based testing is often used in early sub-phases of testing and can be performed by either professional testers or by the developers themselves.

[1] A **Markov chain** is a mathematical system that undergoes transitions from one state to another, between a finite or countable probabilistic number of possible states.

5 Different types of testing

These short examples are only few of the existing types of testing and shall only give a certain impression on what different approaches and types there are to test different aspect of a software system.

Regression Testing

The idea behind regression testing is simply to repeat the tests that have been successful before in case there have been any changes to the code. This ensures that recent changes do not break functionality. You do not want to fix one thing just to realize that you have broken another thing.

Usability Testing

This is a technique to determine how easy the software can be operated. Usability Testing determines user satisfaction, user learning time and the users time effort to complete a task.

Performance Testing

In general, performance testing is performed to determine how fast some aspect of a system performs under a particular workload. Testing techniques that derive from performance testing are load testing, stress testing, and spike testing.

- Load testing is conducted to check whether the system is capable of handling an anticipated workload.
- Stress testing is conducted to check the systems capability beyond the anticipated workload
- Spike testing is conducted to check the systems stability when the load is suddenly increased for a short duration.

Beta Testing

Beta testing takes place when the developing and “normal testing” is essentially completed. Before the final release a group of so called *beta testers* will use the software in an environment similar the real environment. These *beta testers* are people that would otherwise be real customers. This testing technique is performed not by professional testers or developers but by end-users for a period of time to find final bugs and problems. The beta testers so to speak try out the software and help software development organizations improve their software quality.

6 Conclusion

In this paper we gave a short introduction into the topic of software testing and its importance. This aggregation is based on the sixth chapter of Tian's book on Software Quality Engineering [TIAN05].

As shown systematic testing is essential in every software development process in order to ensure software quality. It will not always be possible to do exhaustive testing, meaning testing every possible input for a program. This makes it even more necessary to be aware of the different testing methods and techniques.

As one can see, different techniques serve different purposes. There is not one perfect testing technique that finds all the faults that might be injected into the software. The best approach is to combine different techniques to benefit from them all.

In this paper we have described some of the more commonly known techniques. We also discussed some of the important questions, when it comes to testing: What to test? When to test? When to stop testing?

Finally we would say that time is worthwhile spend when software is tested thoroughly instead of having to deal with immense cost that result from having to fix faults after the software has been released.

7 List of figures

8 Literature

[TIAN05]	Tian, Jeff: Software Quality Engineering – Testing, Quality Assurance, and Quantifiable Improvement, 2005, ISBN 0-471-71345-7
[ISO/IEC25010]	ISO/IEC 25010:2011- Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models http://www.iso.org/iso/iso_catalogue/catalogue_tc ISO/IEC JTC 1/SC 7
[BECK99]	Beck, Kent: Extreme Programming Explained, 1999, ISBN 0201616416 http://software2012team23.googlecode.com/git-history/5127389d21813c2bd955c53999f66cede994578b/docs/literature/Extreme_Programming_Explained_Kent_Beck_1999.pdf
[MUS93]	Musa, John D.: Operational Profiles in Software-Reliability Engineering, March 1993, IEEE Computer Society Press Los Alamitos, CA, USA