Software Engineering

Lecture 10

Software Testing

Dr. Obuhuma James

Description

This topic covers software testing as a critical part of Software Engineering. Software testing begins earlier during requirement gathering through analysis, design, and implementation. On its own, testing stands alone as a phase. This topic hence explores the various types of testing and why and how they are applicable in Software Engineering. The concept of Test-driven Development is also covered. Finally, the topic also outlines when to terminate software testing.

Learning Outcomes

By the end of this topic, you will be able to:

- Discuss the concept of software testing as applied in Software Engineering.
- Differentiate among the various types of software testing.
- Apply software testing in real life.

Overview of Software Testing

As part of system development, a series of software testing activities are carried out. According to Sommerville [1], software testing aims to show that a program does what it is intended to do. Furthermore, testing aids in the discovery of defects in the software before it is put into use.

Software testing is incorporated in the various phases of software development. Thus, some element of testing features all the way from requirements engineering to software maintenance. Each type of testing plays its own role at the given point in time. This will be covered at length in this topic.

Software Testing

Software testing could be viewed as a process of evaluating a system or its components with the aim of finding out whether it satisfies the intended requirements or not. It thus involves subjecting a software to artificial data as you examine the behaviour. The results of testing are examined to point out any errors, anomalies or information about the nonfunctional attributes of the program. In a real case scenario, testing targets at revealing the presence of errors and not their absence. It is worth noting that testing is entranced in a more general verification and validation process that includes static validation methodologies [1].

Ideally, proper software testing result to a reduction in the cost and time for rework and production of error-free software. Obviously, an error-free software will in return immensely reduce maintenance cost for the software.

The earlier and the more the incorporation of testing in the system development life cycle, the better. Hence, software testing should run right from the requirements gathering phase all the way to deployment and maintenance. It is however worth noting that the software development methodology used dictates the level of involvement of software testing. For instance, since the agile methodology is more flexible by allowing backtracking to accommodate changes and heavily relies on incremental development, it makes it easier to incorporate testing throughout the process. In contrary, the plan-driven methodology, uses a rigid approach that does not accommodate backtracking. Thus, in a real case scenario, testing has to occur exactly where the phase falls. Either way, the forms of testing end up varying based on the phase in which it is done, such that,

- The analysis and verification of requirements during the requirement gathering phase can be considered as the first level of testing.
- The review of designs in the design phase is the second level of testing.
- The third level of testing occurs during software implementation where developers test the various units and components as they develop the code.
- The fourth and maybe final level of testing occur upon completion of implementation as the system gets ready for deployment. The testing stretches all the ways to the lifetime of the system as it undergoes maintenance.

Since testing is done at different stages of the software development process, different professionals may be involved including dedicated software testers, software developers, project managers or leads, end users, software vendors among others. In most cases, companies approach software testing in different perspectives leading to varied designations for software testing personnel based on experience and knowledge. These include software tester, software quality assurance engineer, quality assurance analyst roles among other roles.

Software Validation Vs Verification

According to Sommerville [1], part of the main goals for software testing includes

1. Demonstrate that the software meets its requirements. This is critical for both the developer and customer.
2. Bring out scenarios when software behaviour is incorrect, undesirable or isn't conforming to its specification. This is a defect testing scenario.

Software testing thus involves two main aspects, namely, validation and verification. Both are critical towards ensuring suitability of the given software.

Software Validation is a process that involves examination of whether or not a given software satisfies user requirements. It addresses the first goal of software testing such that the system is expected to perform correctly given a set of test cases formulated from user requirements [1]. Software validation answers the question "Are we building the right product?". A successful validation testing should show that the system is operating as intended. A successful test for the second goal is a test that makes the system to function incorrectly with an aim of exposing defects in the system. This makes the test to be referred to as defect testing. Figure 1 shows an input-output model for program testing as used in validation and defect testing.

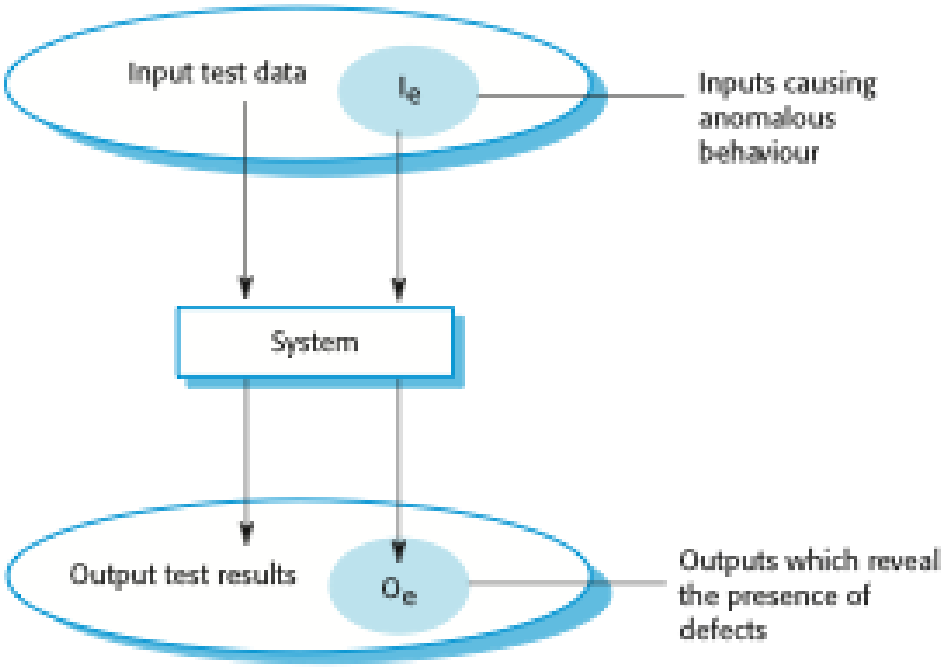

Figure 1. Input-Output Model for Program Testing [1]

Software verification on the other hand confirms if the software under construction meets business requirements and is being developed in adherence to proper design specifications and methodologies. Software verification hence answers the question “Are we building the product, right?”.

According to Sommerville [1], the main aim of software validation and verification (V & V) is to establish confidence in the fitness of the system to its purpose. This fully depends on the purpose of the system, user expectations and the marketing environment.

Software Inspection

According to Sommerville [1], software inspection analyses static system representation to discover problems. The checks are inclined towards conformance with specification and not conformance with real customer requirements. This is in contrary to software testing which inclines towards checks on product behaviour. Thus, software inspection and software testing complement one another without opposing verification techniques. The V & V process should apply both software testing and inspection. It is worth noting that

- Software inspection does not check non-functional characteristics like performance, usability among others.
- Errors can mask or hide other errors during testing. Luckily, since inspection is a static process, the interaction between errors is not a matter of concern.
- Software inspection can span beyond just searching for defects to taking into account broader quality attributes of a program, such as compliance with standards, portability, maintainability, among others.

Software Testing Model

Figure 2 show the software testing model that begins from formulation of test cases and end with test reports. The test cases are evaluated based on the data prepared for the purpose.

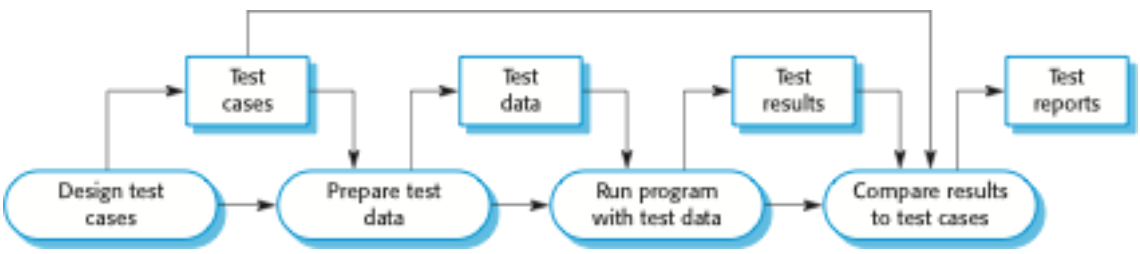

Figure 2. Software Testing Model [1]

Software testing can be carried out manually or automated using special testing tools. Manual testing requires no tools. In this case, test cases are prepared targeting different aspects of the software's code. Tests are then executed with reports analysed to ascertain whether they pass or fail the test as specified in the test cases. Table 1 shows the structure of a test case. Manual testing comes with a number of drawbacks, including being resource consuming, testers need to be keen on whether or not right test cases are used. The limitations of the manual testing approach are overcome by the use of automated testing approaches that are carried out with the aid of automated testing tools.

Table 1. Sample Test Case with Results for a Login Process

Step #	Step Details	Expected Results	Actual Results	Pass / Fail / Not executed / Suspended
1	Navigate to http://xxx.xxx.xxx	• Site should open the login page or provide a link to the login page	Opened login page	Pass
2	Enter Userid & Password	• Credential can be entered	Login credentials capture ok	Pass
3	Enter correct Userid & wrong Password then click the login button	• Error message displayed	Only valid credentials accepted	Pass
4	Enter wrong Userid & correct Password then click the login button	• Error message displayed	No error pops out despite a failed login	Fail
5	Enter correct Userid & correct Password then click the login button	• Successful login. • Land on the homepage (my courses). • Session created	Login successful and lands on homepage	Pass

Testing Approaches

There are two broad approaches used in software testing, namely, functionality testing and implementation testing. Black-box testing is hence considered whenever testing is done without taking actual implementation into consideration. This kind of testing is also known as behavioural testing. The tester uses a set of input data and respective expected output such that the program is deemed to be ok or not ok depending on the output. Thus, the tester, normally a testing engineer or end user, does not know the design and structure of the code during testing. Figure 3 shows the logic of the black-box testing. Black-box testing techniques

include boundary values, equivalence class, cause-effect graphing, pair-wise testing, and state-based testing among others.

Figure 3. Black-box Testing
This Photo by Unknown Author is licensed under CC BY-SA

On the other hand, white-box testing is considered in cases where functionality is tested together with an analysis of how it is implemented. This kind of testing is also known as structural testing. It aims at improving code efficiency and/or program structure. Thus, the tester, normally the programmer, knows the design and structure of the code during testing. Figure 4 shows the logic of the white-box testing. White-box testing techniques include control-flow and data-flow testing.

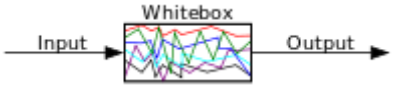

Figure 4. Black-box Testing
This Photo by Unknown Author is licensed under CC BY-SA

Figure 5 show a gray-box testing that is a result of carrying out both a black-box and white-box testing.

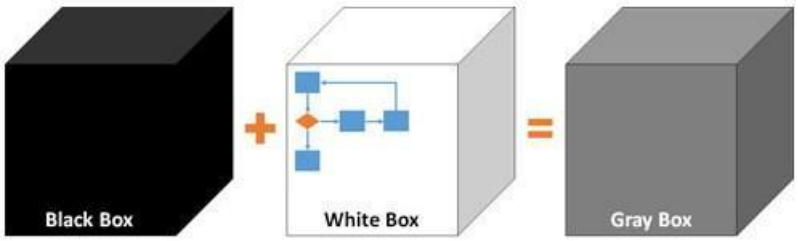

Figure 5. Gray-box Testing
This Photo by Unknown Author is licensed under CC BY-SA-NC

Software Testing Stages

Software testing can be viewed from three testing stages, namely, development testing, release testing, and user testing.

1. Development testing is carried out by the development team. The prime focus for this testing stage is to discover bugs and defects during development. Three kinds of testing are categorized under the development testing stages. These are unit testing, component testing, and system testing.
    a) Unit testing involves testing of individual program units or object classes with a focus on functionality of the objects and methods.
    b) Component testing involves testing of a full component that is an integration of several individual units. The focus is on testing of component interfaces.
    c) System testing involves testing of the whole system where a system can be viewed as an integration of several related components. The focus is on testing of component interactions.
2. Release testing entails testing a particular release of a system intended for use outside of the development team [1]. Release testing aims at convincing the supplier of the system about its goodness. Thus, the release team has to prove that the system meets its specified functionality, performance and dependability without failing during its normal use. It is normally done as a black-box testing process. Performance testing that entails testing of developing properties of a system like system performance and reliability is part of release testing. Release testing is a form of system testing with the following differences:
    a) The test team is separate from the development team, and it should not have been involved in the system development process.
    b) System testing is done by the development team with a focus on discovering bugs in the system (defect testing) while release testing checks that the system meets its requirements and is good enough for external use (validation testing) [1].
3. User or Customer testing entails testing the system using input data. It is critical to do user testing even if system testing and release testing are done comprehensively. This is the case since user's working environments have a major effect on the reliability, performance, usability and robustness of a system which cannot be replicated in a testing environment [1]. Three kinds of testing are categorized under the user testing stages. These are alpha testing, beta testing, and acceptance testing.
    a) Alpha testing involves software users working with the development team to test the software at the developer's site.

b) Beta testing involves availing a release of the software to users to allow them to experiment and to raise any discovered problems with system developers.

c) Acceptance testing lets customers test a system to decide whether or not it is ready to be accepted from the system developers and deployed in the customer environment. Figure 6 shows a summarized guideline for the acceptance testing process.

Figure 6. Acceptance Testing Process [1]

It is worth noting that based on agile methodologies, acceptance testing is not treated as a separate phase since the user or customer is always part of the development process and is always responsible for decision making on system acceptability at every stage. This is however the exact inverse for plan-driven methodologies as discussed in earlier topics.

Test-driven Development

According to Sommerville [1], Test-driven development (TDD) is a program development approach that interleaves testing and code development. The approach works in such a manner that entails writing of tests before code and that passing the tests becomes a crucial determinant for progression.

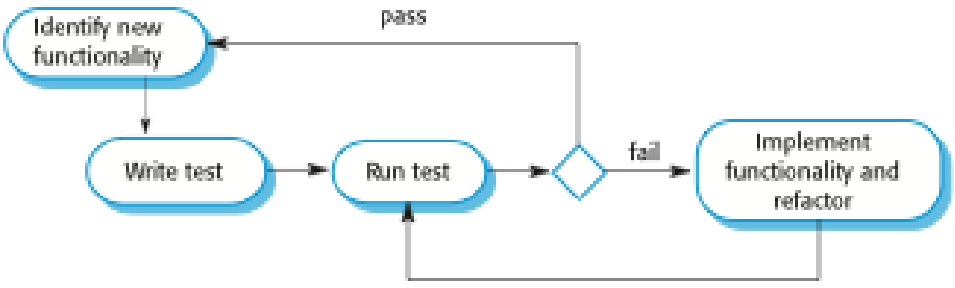

Figure 7. Test-driven Development Process [1]

Based on Figure 7, the TDD approach requires for incremental code development that is interleaved with tests per increment. This ends up with a number of benefits including better

code coverage, development of regression tests, simplified debugging, and enhancement in system documentation.

Termination of Testing

Determination of an ideal stopping time for testing is not easy. In real sense, testing never ends since at no point will it be declared that a given software is 100% tested. Hence, the following may act as guidelines for deciding when to make a stop in software testing:

1. Upon meeting the set testing deadlines.
2. Upon completion of execution of predefine test cases.
3. Achievement of functional and code evaluation coverage to a given point.
4. Decline in the rate of errors and bugs to an acceptable level with no more high priority rated bugs emergence.
5. Based on management decisions.

Summary

The topic has explored software testing to a wider extend by exploring the foundational concepts of software testing, software testing approaches, and software testing stages. The need and differences between software validation and verification is discussed. Similarly, the relationships and differences between software testing and software inspection are discussed. Finally, the concept of Test-driven Development and when to terminate software testing is brought out.

Check Points

1. Discuss the software testing process as used in Software Engineering.
2. Differentiate between software validation and verification as applied to software testing.
3. Describe the relationship between software testing and software inspection.
4. State and explain the software testing stages with their respective activities.
5. Describe the Test-driven Development process as applied in software development and testing.
6. State and explain the various triggers for termination of software testing.

Core Textbooks

1. Sommerville, I., Software Engineering, 10th Edition, Addison Wesley, 2016.

Other Resources

References

[1] Sommerville, I., Software Engineering, 10th Edition, Addison Wesley, 2016.

[2] Pries, K. H., & Quigley, J. M. (2010). Scrum project management. CRC press.

[3] Schwaber, K. (2004). Agile project management with Scrum. Microsoft press.

[4] Gotterbarn, D., Miller, K., & Rogerson, S. (1997). Software engineering code of ethics. Communications of the ACM, 40(11), 110-118.

[5] Pressman, R. S. (2005). Software engineering: a practitioner's approach. Palgrave macmillan.

[6] Kendall, K. E. and Kendall, J. E., Systems Analysis and Design, 8th Edition, Pearson, 2011.