A

Seminar report

On

Software Testing

Submitted in partial fulfillment of the requirement for the award of degree
Of Computer Science

SUBMITTED TO:

www.studymafia.org

SUBMITTED BY:

www.studymafia.org

Preface

I have made this report file on the topic **Software Testing;** I have tried my best to elucidate all the relevant detail to the topic to be included in the report. While in the beginning I have tried to give a general view about this topic.

My efforts and wholehearted co-corporation of each and everyone has ended on a successful note. I express my sincere gratitude to ……..who assisting me throughout the prepration of this topic. I thank him for providing me the reinforcement, confidence and most importantly the track for the topic whenever I needed it.

Acknowledgement

I would like to thank respected Mr…….. and Mr. ……..for giving me such a wonderful opportunity to expand my knowledge for my own branch and giving me guidelines to present a seminar report. It helped me a lot to realize of what we study for.

Secondly, I would like to thank my parents who patiently helped me as i went through my work and helped to modify and eliminate some of the irrelevant or un-necessary stuffs.

Thirdly, I would like to thank my friends who helped me to make my work more organized and well-stacked till the end.

Next, I would thank Microsoft for developing such a wonderful tool like MS Word. It helped my work a lot to remain error-free.

Last but clearly not the least, I would thank The Almighty for giving me strength to complete my report on time.

CONTENTS

INTRODUCTION	3
PROBLEM ARISE IN SOFTWARE	4
HISTORY	5
OBJECTIVE OF SOFTWARE TESTING	6
TYPES OF TESTING	7
WHITE BOX TESTING	8-9
BLACK BOX TESTING	10-11
LEVELS OF TESTING	12
UNIT TESTING	13
INTEGRATION TESTING	14
SYSTEM TESTING	15
ACCEPTANCE TESTING	16
REGRESSION TESTING	17
ALPHA AND BETA TESTING	18
TESTING CYCLE	19-20
REFERENCES	21

INTRODUCTION

Testing is the last step in the software life cycle. Time pressure is well known and increasing because too many defects are found late and have to be repaired. This seminar shows you how to cope with this situation. Early test planning and the use of reviews achieve a high degree of preventive defect removal. Still, at the end you have to execute the tests, in order to measure the final product quality. The ultimate aim, however, is that defects show up during test preparation rather than test execution.

Testing can never completely identify all the defects within software. Instead, it furnishes a *criticism* or *comparison* that compares the state and behavior of the product against oracles—principles or mechanisms by which someone might recognize a problem. These oracles may include (but are not limited to) specifications, contracts comparable products, past versions of the same product, inferences about intended or expected purpose, user or customer expectations, relevant standards, applicable laws, or other criteria.

Every software product has a target audience. For example, the audience for video game software is completely different from banking software. Therefore, when an organization develops or otherwise invests in a software product, it can assess whether the software product will be acceptable to its end users, its target audience, its purchasers, and other stakeholders. **Software testing** is the process of attempting to make this assessment.

A study conducted by NIST in 2002 reports that software bugs cost the U.S. economy $59.5 billion annually. More than a third of this cost could be avoided if better software testing was performed.

PROBLEM ARISES IN SOFTWARE:-

- Incorrect calculation
- Incorrect data edits & ineffective data edits
- Incorrect matching and merging of data
- Data searches that yields incorrect results
- Incorrect processing of data relationship
- Incorrect coding / implementation of business rules
- Inadequate software performance
- Confusing or misleading data
- Software usability by end users & Obsolete Software
- Inconsistent processing
- Unreliable results or performance
- Inadequate support of business needs
- Incorrect or inadequate interfaces with other systems
- Inadequate performance and security controls
- Incorrect file handling

HISTORY

The separation of debugging from testing was initially introduced by Glenford J. Myers in 1979. Although his attention was on breakage testing ("a successful test is one that finds a bug") it illustrated the desire of the software engineering community to separate fundamental development activities, such as debugging, from that of verification. Dave Gelperin and William C. Hetzel classified in 1988 the phases and goals in software testing in the following stages:

- Until 1956 - Debugging oriented
- 1957–1978 - Demonstration oriented
- 1979–1982 - Destruction oriented
- 1983–1987 - Evaluation oriented
- 1988–2000 - Prevention oriented

OBJECTIVES OF TESTING

- Executing a program with the intent of finding an *error*.
- To check if the system meets the requirements and be executed successfully in the Intended environment.
- To check if the system is " Fit for purpose".
- To check if the system does what it is expected to do.
- Executing a program with the intent of finding an *error*.
- To check if the system meets the requirements and be executed successfully in the Intended environment.
- To check if the system is " Fit for purpose".
- To check if the system does what it is expected to do.

TYPES OF TESTING METHODOLOGIES

Software testing methods are traditionally divided into two parts. These are :-

1. White box testing
2. Black-box testing.

These two approaches are used to describe the point of view that a test engineer takes when designing test cases.

WHITE BOX TESTING

White box testing is when the tester has access to the internal data structures and algorithms including the code that implement these.

- All independent paths within a module have been exercised at least once
- Exercise all logical decisions on their *true* and *false* sides
- Execute all loops at their boundaries and within their operational bounds
- Exercise internal data structures to ensure their validity

Types of white box testing

The following types of white box testing exist:

- **Statement Coverage** – execute all statements at least once
- **Decision Coverage** – execute each decision direction at least once
- **Condition Coverage** – execute each decision with all possible outcomes at least once
- **Decision / Condition coverage** – execute all possible combinations of condition outcomes in each decision.

- **Multiple condition Coverage** – Invokes each point of entry at least once.

Test coverage

White box testing methods can also be used to evaluate the completeness of a test suite that was created with black box testing methods. This allows the software team to examine parts of a system that are rarely tested and ensures that the most important function points have been tested.

Two common forms of code coverage are:

- *Function coverage*, which reports on functions executed
- *Statement coverage*, which reports on the number of lines executed to complete the test

They both return a code coverage metric, measured as a percentage.

Black box testing

Black box testing treats the software as a "black box"—without any knowledge of internal implementation. Black box testing methods include: equivalence partitioning, boundary value analysis, all-pairs testing, fuzz testing, model-based testing, exploratory testing and specification-based testing.

- Based on requirements and functionality
- Not based on any knowledge of internal design or code
- Covers all combined parts of a system
- Tests are data driven

Specification-based testing: Specification-based testing aims to test the functionality of software according to the applicable requirements. Thus, the tester

inputs data into, and only sees the output from, the test object. This level of testing usually requires thorough test cases to be provided to the tester, who then can simply verify that for a given input, the output value (or behavior), either "is" or "is not" the same as the expected value specified in the test case.

Specification-based testing is necessary, but it is insufficient to guard against certain risks.

Advantages and disadvantages: The black box tester has no "bonds" with the code, and a tester's perception is very simple: a code *must* have bugs. Using the principle, "Ask and you shall receive," black box testers find bugs where programmers do not. On the other hand, black box testing has been said to be "like a walk in a dark labyrinth without a flashlight," because the tester doesn't know how the software being tested was actually constructed.

As a result, there are situations when

(1) a tester writes many test cases to check something that could have been tested by only one test case,

(2) some parts of the back-end are not tested at all.

Therefore, black box testing has the advantage of "an unaffiliated opinion", on the one hand, and the disadvantage of "blind exploring", on the other.

LEVELS OF TESTING

Tests are frequently grouped by where they are added in the software development process, or by the level of specificity of the test. The main levels during the development process as defined by the SWEBOK guide are unit-, integration-, and system testing that are distinguished by the test target without implying a specific process model. Other test levels are classified by the testing objective.

The following levels of testing are:-

1. Unit testing
2. Integration testing
3. System testing
4. Acceptance testing
5. Regression testing

UNIT TESTING

Unit testing, also known as *component testing*, refers to tests that verify the functionality of a specific section of code, usually at the function level. In an object-oriented environment, this is usually at the class level, and the minimal unit tests include the constructors and destructors.

These types of tests are usually written by developers as they work on code (white-box style), to ensure that the specific function is working as expected. One function might have multiple tests, to catch corner cases or other branches in the code. Unit testing alone cannot verify the functionality of a piece of software, but rather is used to assure that the building blocks the software uses work independently of each other.

Tests the smallest individually executable code units. Usually done by programmers. Test cases might be selected based on code, specification, intuition, etc.

Tools:

- Test driver/harness
- Code coverage analyzer
- Automatic test case generator

INTEGRATION TESTING:-

Integration testing is any type of software testing that seeks to verify the interfaces between components against a software design. Software components may be integrated in an iterative way or all together ("big bang"). Normally the

former is considered a better practice since it allows interface issues to be localised more quickly and fixed.

Integration testing works to expose defects in the interfaces and interaction between integrated components (modules). Progressively larger groups of tested software components corresponding to elements of the architectural design are integrated and tested until the software works as a system.

How are units integrated? What are the implications of this order?

- Top-down => need stubs; top-level tested repeatedly.
- Bottom-up => need drivers; bottom-levels tested repeatedly.
- Critical units first => stubs & drivers needed; critical units tested repeatedly.

Potential Problems:

- Inadequate unit testing.
- Inadequate planning & organization for integration testing.
- Inadequate documentation and testing of externally-supplied components.

SYSTEM TESTING

System testing tests a completely integrated system to verify that it meets its requirements.

Realities of System Testing

- Not all problems will be found no matter how thorough or systematic the testing.
- Testing resources (staff, time, tools, labs) are limited.

- Specifications are frequently unclear/ambiguous and changing (and not necessarily complete and up-to-date).
- Systems are almost always too large to permit test cases to be selected based on code characteristics.

REGRESSION TESTING

Regression testing focuses on finding defects after a major code change has occurred. Specifically, it seeks to uncover software regressions, or old bugs that have come back. Such regressions occur whenever software functionality that was previously working correctly stops working as intended. Typically, regressions occur as an unintended consequence of program changes, when the newly developed part of the software collides with the previously existing code. Common methods of regression testing include re-running previously run tests and checking whether previously fixed faults have re-emerged. The depth of testing depends on the phase in the release process and the risk of the added features. They can either be complete, for changes added late in the release or deemed to be risky, to very shallow, consisting of positive tests on each feature, if the changes are early in the release or deemed to be of low risk.

ACCEPTANCE TESTING

Acceptance testing can mean one of two things:

1. smoke test is used as an acceptance test prior to introducing a new build to the main testing process, i.e. before integration or regression.
2. Acceptance testing performed by the customer, often in their lab environment on their own hardware, is known as user acceptance testing (UAT). Acceptance testing may be performed as part of the hand-off process between any two phases of development.

ALPHA TESTING

Alpha testing is simulated or actual operational testing by potential users/customers or an independent test team at the developers' site. Alpha testing is often employed for off-the-shelf software as a form of internal acceptance testing, before the software goes to beta testing.

BETA TESTING

Beta testing comes after alpha testing and can be considered a form of external user acceptance testing. Versions of the software, known as beta versions, are released to a limited audience outside of the programming team. The software is released to groups of people so that further testing can ensure the product has few faults or bugs. Sometimes, beta versions are made available to the open public to increase the feedback field to a maximal number of future users

TESTING CYCLE

Although variations exist between organizations, there is a typical cycle for testing. The sample below is common among organizations employing the Waterfall development model.

- **Requirements analysis**: Testing should begin in the requirements phase of the software development life cycle. During the design phase, testers work with developers in determining what aspects of a design are testable and with what parameters those tests work.
- **Test planning**: Test strategy, test plan, testbed creation. Since many activities will be carried out during testing, a plan is needed.
- **Test development**: Test procedures, test scenarios, test cases, test datasets, test scripts to use in testing software.
- **Test execution**: Testers execute the software based on the plans and test documents then report any errors found to the development team.
- **Test reporting**: Once testing is completed, testers generate metrics and make final reports on their test effort and whether or not the software tested is ready for release.
- **Test result analysis**: Or Defect Analysis, is done by the development team usually along with the client, in order to decide what defects should be assigned, fixed, rejected (i.e. found software working properly) or deferred to be dealt with later.
- **Defect Retesting**: Once a defect has been dealt with by the development team, it is retested by the testing team. AKA Resolution testing.
- **Regression testing**: It is common to have a small test program built of a subset of tests, for each integration of new, modified, or fixed software, in order to

ensure that the latest delivery has not ruined anything, and that the software product as a whole is still working correctly.

- **Test Closure**: Once the test meets the exit criteria, the activities such as capturing the key outputs, lessons learned, results, logs, documents related to the project are archived and used as a reference for future projects.

REFERENCES

1. McConnell, Steve (2004). *Code Complete* (2nd ed.). Microsoft Press. pp. 29. ISBN 0-7356-1967-0.
2. Principle 2, Section 1.3, Certified Tester Foundation Level Syllabus, International Software Testing Qualifications Board
3. Tran, Eushiuan (1999). "Verification/Validation/Certification". In Koopman, P.. *Topics in Dependable Embedded Systems*. USA: Carnegie Mellon University. http://www.ece.cmu.edu/~koopman/des_s99/verification/index.html. Retrieved 2008-01-13.
4. see D. Gelperin and W.C. Hetzel
5. Introduction, Code Coverage Analysis, Steve Cornett
6. Laycock, G. T. (1993) (PostScript). *The Theory and Practice of Specification Based Software Testing*. Dept of Computer Science, Sheffield University, UK. http://www.mcs.le.ac.uk/people/gtl1/thesis.ps.gz. Retrieved 2008-02-13.
7. Patton, Ron. *Software Testing*.
8. Binder, Robert V. (1999). *Testing Object-Oriented Systems: Objects, Patterns, and Tools*. Addison-Wesley Professional. p. 45. ISBN 0-201-80938-9.
9. Beizer, Boris (1990). *Software Testing Techniques* (Second ed.). New York: Van Nostrand Reinhold. pp. 21,430. ISBN 0-442-20672-0.
10. IEEE (1990). *IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries*. New York: IEEE. ISBN 1559370793.