www.lpude.in

DIRECTORATE OF DISTANCE EDUCATION

SOFTWARE TESTING AND QUALITY ASSURANCE

Produced & Printed by
EXCEL BOOKS PRIVATE LIMITED
A-45, Naraina, Phase-I,
New Delhi-110028
for
Directorate of Distance Education
Lovely Professional University
Phagwara

Directorate of Distance Education

LPU is reaching out to the masses by providing an intellectual learning environment that is academically rich with the most affordable fee structure. Supported by the largest University[1] in the country, LPU, the Directorate of Distance Education (DDE) is bridging the gap between education and the education seekers at a fast pace, through the usage of technology which significantly extends the reach and quality of education. DDE aims at making Distance Education a credible and valued mode of learning by providing education without a compromise.

DDE is a young and dynamic wing of the University, filled with energy, enthusiasm, compassion and concern. Its team strives hard to meet the demands of the industry, to ensure quality in curriculum, teaching methodology, examination and evaluation system, and to provide the best of student services to its students. DDE is proud of its values, by virtue of which, it ensures to make an impact on the education system and its learners.

Through affordable education, online resources and a network of Study Centres, DDE intends to reach the unreached.

[1] in terms of no. of students in a single campus

SYLLABUS

Software Testing and Quality Assurance

Objectives: The objective of this course is to impart understanding of techniques for software testing and quality assurance. To help students to develop skills that will enable them to construct software of high quality - software that is reliable, and that is reasonably easy to understand, modify and maintain.

S. No.	Description
1.	***Introduction to Software Testing:*** Introduction, Definition of a Bug, Role of a Software Tester, Software Development Model, Software Testing Axioms, Software Testing Terms and Definitions.
2.	***Fundamentals of Software Testing:*** Testing Strategies and Techniques, Structural and Functional testing, Static Black Box and Dynamic Black Box Testing Techniques.
3.	***White Box Testing:*** Static White Box Testing, Dynamic White Box Testing.
4.	***Special Types of Testing:*** Configuration Testing, Compatibility Testing, Graphical User Interface Testing.
5.	***Documentation and Security Testing:*** Documentation Testing, Security Testing.
8.	***Web site Testing:*** Web Page Fundamentals, Black Box Testing, White Box Testing and Gray Box Testing, Configuration and Compatibility Testing.
9.	***Testing Tools:*** Benefits of Automation Testing, Random Testing, Bug Bashes and Beta Testing.
8.	***Test Planning:*** Test Planning, Test Cases, Bug life cycle.
9.	***Software Quality Assurance:*** Definition of Quality, Testing and Quality Assurance at Workplace, Test Management and Organizational Structure, Software Quality Assurance Metrics, Quality Management in IT.
10.	***Organizational Structure:*** CMM (Capability Maturity Model), ISO 9000, Software Engineering Standards.

CONTENTS

Unit 1: Introduction to Software Testing

CONTENTS

Objectives

After studying this unit, you will be able to:

- State the importance of testing in the software development life cycle
- Describe the evolution and types of software testing
- Define a bug and illustrate the reason of occurrence of a bug
- Describe the software development models

Introduction

The success of any software product or application is greatly dependent on its quality. Today, testing is seen as the best way to ensure the quality of any product. Quality testing can greatly reduce the cascading impact of rework of projects, which have the capability of increasing the budgets and delaying the schedule. The need for testing is increasing, as businesses face pressure to develop sophisticated applications in shorter timeframes. Testing is a method of investigation conducted to assess the quality of the software product or service. It is also the process of checking the correctness of a product and assessing how well it works.

The process of testing identifies the defects in a product by following a method of comparison, where the behavior and the state of a particular product is compared against a set of standards which include specifications, contracts, and past versions of the product. Software testing is an incremental and

iterative process to detect a mismatch, a defect or an error. As pointed by Myers, "Testing is a process of executing a program with the intent of finding errors".

According to IEEE 83a, "Software testing is the process of exercising or evaluating a system or system component by manual or automated means to verify that it satisfies specified requirements."

1.1 Software Testing

Software testing is an integral part of the software development life cycle which identifies the defects, flaws or the errors in the application. It is incremental and iterative in nature. The goal of testing as described by Millers states that, "The general aim of testing is to affirm the quality of software systems by systematically exercising the software in carefully controlled circumstances". Let us now list out the objectives of software testing:

1. It ensures if the solutions meet the business requirements, thereby enhancing customer confidence.
2. It catches the bugs, errors and defects.
3. It ensures if the system is stable and ready for use.
4. It identifies the areas of weakness in an application or product.
5. It establishes the degree of quality.
6. It determines user acceptability

The scope of testing can be comprehensive and could examine components like business requirements, design requirements, programmer's code, hardware configuration, and systems administration standards and constraints. The ambit of testing could also involve testing with respect to industry standards and professional best practices. Testing thus provides an opportunity to validate and verify all aspects of software engineering.

Thus, software testing brings many benefits to an organization. It saves time and money, since defects are identified early. Testing ensures that the product is stable with less downtime, thereby gaining customer satisfaction.

Did you know? In the year 2003, social security checks for 50,000 people were mailed by the U.S. Treasury Department without any beneficiary name. It was later found out that the missing names were due to a software program maintenance error.

1.1.1 Evolution of Software Testing

During the early days, software was developed by small groups of people, but had to be maintained by a different set of people. It was at this time that people who were maintaining the software had nightmarish experiences, as the software would throw up a lot of errors. Due to this, it was difficult to handle large projects and thus, tasks were not completed on time and within budget. As a result, projects were delayed and some of them were abandoned halfway.

Software testing evolved over time. Advances in software development brought in new changes in the way testing was perceived, which led to systematic improvement in testing.

Deve Gelperin and William C. Hetzel classified software testing based on the goals and phases of testing, which are as follows:

1. ***Until 1956 - Debugging Oriented:*** Until 1956, testing and debugging were not differentiated, and testing connoted debugging.
2. ***1957–1978 - Demonstration Oriented:*** During this period, the goal of testing was to make sure that the software satisfies its specifications. The period of 70s also witnessed the acceptance of the idea that software could be tested exhaustively.
3. ***1979–1982 - Destruction Oriented:*** During this period, testing grew in importance with contributions from Myers (author of "The Art of Software Testing"). Test cases that had the intent of finding errors were designed and verification and validation activities were initiated.

4. ***1983–1987 - Evaluation Oriented:*** During this period, Guidelines for Lifecycle Validation, Verification and Testing of Computer Software was created by the National Bureau of standards in 1983. This methodology enabled testing to be integrated with analysis, design and implementation of the software lifecycle.
5. ***1988-2000-Prevention Oriented:*** During this period, the goal of testing metamorphed into a mechanism to prevent errors rather than just detecting errors. Hetzel, in 1991, gave the definition of Testing as "Testing is planning, designing, building, maintaining and executing tests and test environments". There was greater emphasis of early test design and a process, consisting of three phases -- test planning, test analysis, and test design evolved.

1.2 Definition of Bug

A bug, also known as a software bug, is an error in a software program that may produce incorrect, undesired result or prevent the program from working correctly. In software testing, a bug not only means an error, but anything that affects the quality of the software program. Software bugs take different names such as – defect, fault, problem, error, incident, anomaly, failure, variance and inconsistency and so on.

The following are certain conditions that result in a bug in a software:

1. If the software does not respond or act in the way as stipulated in the product specification.
2. If the software behaves in a way that is stipulated in an opposite way in the product specification.
3. If the software responds or reacts in a way that is not mentioned in the product specification.
4. If the software does not behave in the mandatory way as expected --perhaps, this might not be mentioned in the product specification.
5. If the software is difficult to understand or has cumbersome steps to follow, or if it is slow in its reaction.

The above rules can be applied to a calculator to understand it better. We are aware that the calculator should perform the operations of addition, subtraction, multiplication and division. If the + key does not work, the bug follows the rule specified in first condition, namely, no response in the way as stipulated in the specification. The calculator is not expected to crash and freeze, and if it does, the bug follows the rule specified in condition 2, that is behaving in the opposite way as stipulated in the specification. If there are additional features found in the calculator, namely a function to find the squares of a number, but is not mentioned in the specification, it falls in the rule specified in the third condition, where there is no mention in the product specification. Let us assume that the battery condition is weak, and at this stage, the calculator does not give the right answers. This condition is a typical situation which follows condition four. Condition 5 brings the customer perspective. It occurs whenever the performance of the software is not in an acceptable way to the user – a sample case where the tester finds the space allotted for the button very small or if the lights are too flashy that the user is unable to see the answer, it follows condition 5. The feedback of the tester becomes important, as it brings out the negatives even before the product reaches the customer.

Example: During the test phase of a mobile phone application, it was detected that when the numerical key, zero, is clicked, the value is not displayed on the screen. This might have occurred due to many reasons- hardware or software related issues. However, though no error occurs in such a condition, the quality parameter of the mobile application is compromised, and hence it is considered as a bug.

1.2.1 Types of Bugs

Software bugs, which occur irrespective of the size of the program, are generally encountered when different groups of developers work to develop a single program. We will now list the common types of bugs and their causes.

Some of the common types of software bugs include the following:

1. Bugs due to conceptual error

Example: Incorrect usage of syntax in the program, misspelled keywords, using wrong or improper design or concept.

2. Math bugs

Example: Divide by zero error, overflow or underflow, lack of precision in arithmetic values due to incorrect rounding or truncation of decimal values.

3. Logical bugs

Example: Infinite loops, infinite recursion, applying wrong logic, incorrect usage of jump or break conditions.

4. Resource bugs

Example: Stack or buffer overflow, access violations, using variables that are not initialized.

5. Co-programming bugs

Example: Concurrency errors, deadlock, race condition.

6. Team working bugs

Example: Out of date comments, non-matching of documentation or files, linking the program to incorrect files.

Reasons for Occurrence of a Bug

Bugs can occur in a program due to many reasons. Some of the common reasons that may cause bugs in software are:

1. ***Human Factor:*** The developer who develops the software is human and there is always a chance that some error, incorrect usage of logic or syntax or improper linking of files might occur. Since the entire software development process involves humans, it is prone to errors.
2. ***Communication Failure:*** The lack of communication or communicating incorrect information during the various phases of software development will also lead to bugs. Lack of communication can happen during various stages of Software Development Life Cycle (Requirements, Design, Development, Testing, and Modification). Many bugs occur due to lack of communication when a developer tries to modify software developed by another developer.

Example: The requirements defined in the initial phase of the Software Development Life Cycle may not be properly communicated to the team that handles the design of the software. The design specifications that reach the development team may be incorrect, which could lead to occurrence of bugs in the software.

3. ***Unrealistic Development Timeframe:*** Sometimes developers work under tight schedules, with limited or insufficient resources, and unrealistic project deadlines. Compromises are made in the requirements or design of the software in order to meet the delivery requirements. Most of the time, it leads to wrong understanding of the requirements, inefficient design, improper development, and inadequate testing.
4. ***Poor Design Logic:*** During complex software development, some amount of R&D and brainstorming has to be done to develop a reliable design. It is also important to choose the right components, products and techniques for software development and testing during the design phase. If there is any lack of proper understanding in the technical feasibility of the components, products, or techniques in the development process, it may cause bugs in the software that is being developed.

5. ***Poor Coding Practices:*** Poor coding practices such as inefficient use of codes, misspelled keywords, or incorrect validation of variables will lead to bugs. Sometimes, faulty tools such as editors, compilers, and debuggers generate wrong codes, which cause errors in the software and it is difficult to debug such errors.
6. ***Lack of Skilled Testing:*** If there are any drawbacks in the testing process of the software, bugs go unnoticed.
7. ***Change Requests:*** If there are last minute changes on requirements, tools, or platform, it could cause errors and can lead to serious problems in the application.


Task

Browse the web and collect information pertaining to some of the famous bugs encountered during a project and the reasons for their occurrence.

http://www.wired.com/software/coolapps/news/2005/11/69355
http://www5.in.tum.de/~huckle/bugse.html
http://www.cds.caltech.edu/conferences/1997/vecs/tutorial/Examples/Cases/failures.htm

1.2.2 Cost of Bugs

Any bug that exists in the software delivered to the customer can lead to huge financial losses. It will bring down the reputation of the organization developing the software. In case the bug causes any damage to life or property of the customer, it can also lead to huge penalty on the organization that developed the software. Bugs that are detected during the testing stage will take some time to get fixed, affecting the timeline and increasing the project cost.

If the bugs are detected and fixed at the earliest in the software development life cycle, it can result in higher return on investment for the organization with respect to time and cost. The cost of fixing a bug differs depending on the development stage at which it is detected. Let us analyze the impact when bugs are detected at different stages of project life cycle.

1. ***Requirement Stage:*** It would cost the organization some time to rewrite the requirements and can be easily detected and fixed.
2. ***Coding Stage:*** It would make the developer spend some time to debug. However, the time required to fix the bug will vary depending on the complexity of the bug. A bug detected by the developer at this stage can be resolved easily, since the developer understands the problem and will be able to fix it.
3. ***Integration Stage:*** It will require more time to be resolved. Since the problem occurs at a higher level, it requires time to check the part of the code or configuration which is wrong and additional time to fix the bug. The developer and other system engineers will be involved in detecting and resolving the bug.
4. ***Testing Stage:*** It will involve the developer, system engineer, and project manager for detecting and resolving the bug. It is an iterative process, which takes a lot of time and effort with respect to man hours and cost in order to detect and fix the bug. Since the bugs have to be tracked and prioritized for debugging, it will increase the project cost in terms of man power and tools required and causes delay in the delivery times as well.
5. ***Production Stage:*** It will take huge time and investment for the organization, as it involves developers, system engineers, project managers, and customers for detecting and resolving the bug. It demands prioritizing and detailed planning when compared to a bug detected in testing stage. Such bugs not only cause a huge loss, but also bring down the reputation of the organization.

A bug found and fixed during the early stages – say, while defining the specification or requirements, might cost the company a very small amount, for example, consider it to be one rupee per bug. If the same bug is found during the coding and testing stage, it costs the company reasonably, for example, 10

to 100 rupees. If the same bug is found by the customer, then the cost would be very high, that is, the cost the company has to pay can be from a few thousands to a few lakhs. Along with it, the reputation of the company will also be damaged.

Caselet

Costly Bug that Caused the Death of Soldiers

The Patriot missile defense system of U.S. Defence forces was first put to use during the Gulf War (1990/1991) as a counter attack against the Iraqi Scud missiles. Though the Patriot is considered to be a very successful anti-missile system, it failed to defend against the enemy missiles attacks several times. A well known incident is the killing of 28 U.S. soldiers in Dhahran, Saudi Arabia. This incident took place since the anti-missile system had failed to defend against the enemy missile attack.

Later it was found that the failure had occurred due to a bug. Due to a simple timing error in the anti-missile system's clock, the tracking system was not accurately tracking the anti-missiles after 14 hours and the missile system was operational in Dhahran for more than 100 hours.

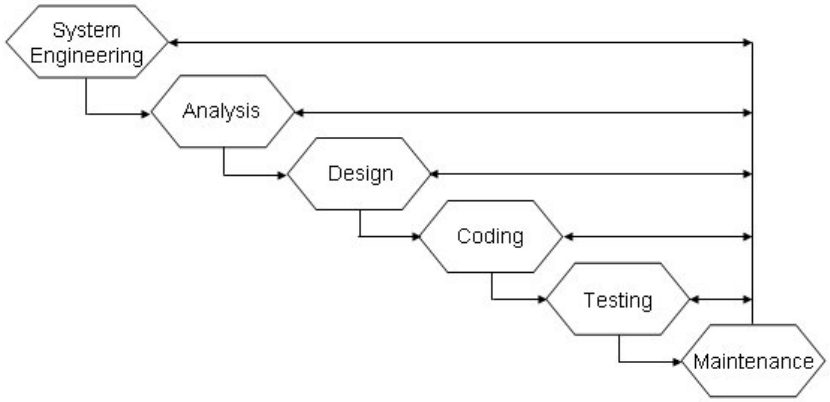
1.3 Software Development Models

A software life cycle development model describes the different phases or activities of a project from its conception. Various software development models are used based on the requirement of the project. We shall now describe some of the most popular and widely used software development models, which include the waterfall model, V model, spiral model, RAD model, prototyping model, and Agile model. According to the nature of the model, the testing approach also differs.

1.3.1 Waterfall Model

This is one of the oldest software lifecycle models. The process starts at the system level and is followed by various phases like the analysis, design, coding, testing and maintenance as depicted in figure 1.1.

Figure 1.1: Waterfall Model



1. ***System Engineering:*** In this phase, system requirements that are essential for the development of the software are defined. These requirements mainly define the software and the hardware requirements relevant for the software development process.
2. ***Analysis:*** In this phase, the developers conduct feasibility studies to define the goals of development. The performance and interfacing requirements for the software are listed out.

3. ***Design:*** In this phase, the requirements described during the analysis phase are defined in terms of the software structure – for example, the database design is completed. These representations help in determining the logical flow of the software and also help in quality assessment. These specifications help the developers to provide inputs during the actual coding process.
4. ***Coding:*** This is the programming phase where the design is developed into a machine-readable form. The developer writes the source code using a software language as per the design specification.
5. ***Testing:*** In this phase, the software is tested against the documented test methods. Testing detects the possible bugs and makes the necessary corrections. These tests also enable the developers to know whether the software is performing according to the requirements.
6. ***Maintenance:*** As new requirements arise, there is a need to upgrade the software. There are instances when problems which need to be solved during the live production environment arise. These are done during the maintenance phase.

The main drawback of this model is that the errors and defects that are present in one phase of the lifecycle are passed on to the other, which results in longer delays and additional costs as the problems need to be solved at each stage.

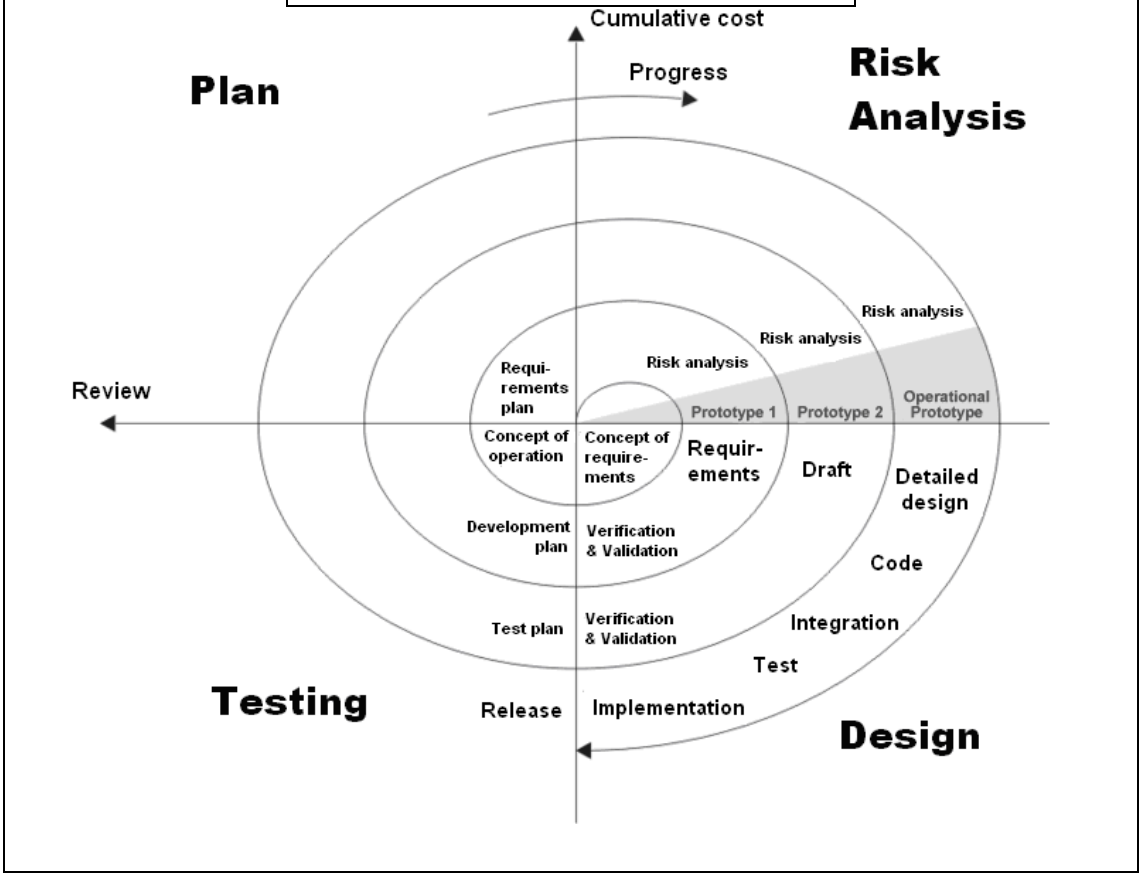
1.3.2 Spiral Model

The drawbacks of the waterfall model are overcome in the spiral model. The model consists of four phases - planning, risk analysis, design engineering and customer evaluation. The four phases are iteratively followed till the problems are rectified. Two to three prototypes are developed before the final product is delivered. Each prototype follows the entire cycle to solve problems. This method of software development enables to understand the problems associated with a particular phase and deals with those problems when the same phase is repeated again. The figure 1.2 is the pictorial representation of spiral model. The various phases involved in each cycle are,

1. ***Plan:*** In this phase, the specifications, objectives, constraints, and alternatives of the project are listed in logical order as per the project requirements. The objectives and specifications are defined in order to decide the strategies to be followed during the project life cycle.
2. ***Risk Analysis:*** This is a very crucial phase of spiral model. During this phase, all the alternatives that are helpful in developing a cost effective project are analyzed and all possible risks involved in the project development are identified.
3. ***Design Engineering:*** This is the phase where the actual development of the software takes place. The software product is developed iteratively and passed on to the next phase.
4. ***Testing:*** In this phase, the customer receives the product and gives comments and suggestions which can help in identifying and resolving potential problems in the developed software. During this cycle all the phases concentrate on the feedback received from the customer and the testing team to resolve the drawbacks and bugs found in each prototype of the product. The main drawback of the model is the amount of time taken to complete the iterations which can increase costs. Testing at the customer's end and fixing of bugs might require higher cost and time.

Figure 1.2 shows the Spiral model of software development.

Figure 1.2: Spiral Model



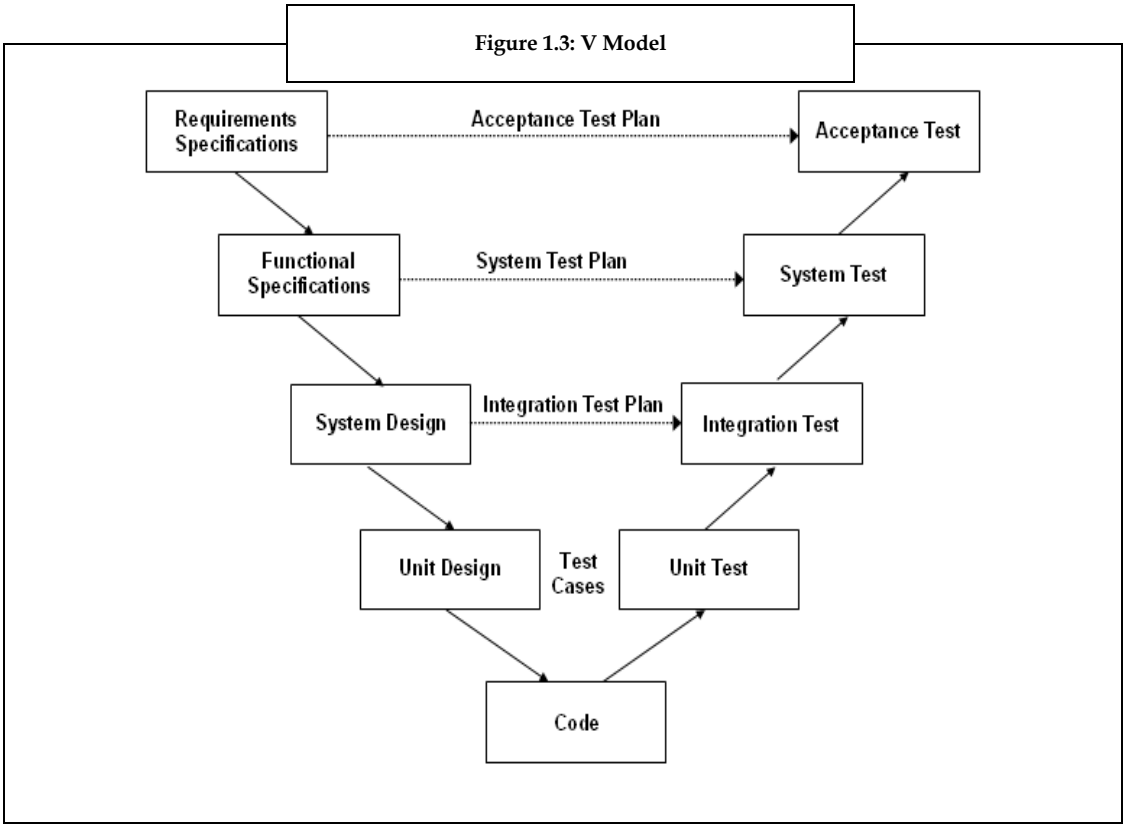
1.3.3 V-Model

The drawbacks of other models are overcome if testing starts at the beginning of the project. V model is a popular method since it incorporates testing into the entire development life cycle. This model ensures proper quality checks throughout the project lifecycle.

The salient aspect of V model is that it portrays distinct testing levels and illustrates how each level addresses a different stage of the lifecycle. The development activities begin with defining the business requirements, where it moves from the high level to low level design, whereas the testing cycle moves from the low level to the higher level. The customer provides the requirement specifications that define the business requirements, which is followed by the functional specification phase. In this phase the process, structural, and event models are developed to understand the requirement specifications carefully. It is very important to carefully define the specification during this stage, since it decides the effectiveness of testability during the test phase, i.e., during system test stage.

Figure 1.3 shows the pictorial representation of the V model. The left half of the V model is the software development phase and the right side of the diagram shows the test phase.

Figure 1.3: V Model



During the unit design stage, the individual programs or modules of the entire software are specified, based on which the test cases for unit testing are developed. Test cases are constructed to check various aspects of the software. These checks are carried out to test the actual program structure or to test whether the software functions as per the specification.

Example: While developing test cases to check the software developed for a calculator, we could check the actual code logic used to perform certain calculations, memory required by each module, techniques followed to link various modules of the software, and the overall efficiency.

Unit testing focuses on the types of faults that occur when writing code. Integration testing focuses on low-level design, especially those errors that occur in interfaces. System test evaluates if the system effectively implements the high level design, specifically the adequacy of performance in a production setting. Acceptance tests are performed by the customers to ensure that the product meets the business requirements. The powerful benefit of this model is that testers are able to verify and validate the specification at an earlier point in the project. This reduces the defects and builds in quality significantly. The only drawback that is seen in this model is that it is not suitable when the requirements are not fully documented and is not applicable to all aspects of development and testing.

1.3.4 Rapid Application Development Model

Rapid Application Development (RAD) is a software development model which is created from the business requirements, project management requirements and software requirements specifications (SRS). In this model, a prototype is created and matched against the requirements. If there is a gap, there is another model created and prototype developed. This model follows a linear sequential software development process, where an extremely short development cycle is adopted and a re-usable component is used for development. When the requirements are well understood and defined, the RAD process enables the development team to develop the final product in a shorter period.

The RAD model consists of the following four phases:

1. ***Requirements Planning:*** During this phase, the project requirements are gathered and the project outline is planned.
2. ***Design Phase:*** The RAD team prepares a design or model (prototype) of the system required. While developing the design or model, all the requirements as well as the changes in the previous phase are listed.
3. ***Construction Phase:*** This is the phase where the various RAD tools are used to develop the first prototype of the model. The prototype developed is based on the design phase of RAD model. The prototype is checked to make sure that it meets the customer requirements. In case, the customer is not satisfied, then changes are made in the model and one more prototype is built. This modification process is carried out until the customer is completely satisfied with the product.
4. ***Testing and Handover Phase:*** The testing and implementation phase relies heavily on the re-use of software components. Since the software components would have been already tested while being developed for other projects, the time spent on testing the software is very less. Once the testing has been completed the software product can be implemented and used by the customer.


Notes

Disadvantages of RAD model

1. It has to be made sure that no reusable component is missing, since it can lead to failure of the entire project.
2. This method is suitable for small projects only and cannot be applied for complex projects.

1.3.5 Agile Model

The Agile software development model is the most popular model used today. The traditional software development models follow either sequential waterfall model or iterative spiral model. Such software models cannot be efficiently adapted to complex software development projects that have continuous and multiple changes. Therefore, the Agile software development model was developed, which responds to changes quickly and smoothly. The control mechanism used in Agile is the feedback from people. The feedback helps to break work into small pieces which is tested and further refined with user input.


Did you Know? The disadvantages that the developers faced in sequential models were overcome by iterative methodologies. However, iterative methodologies still follow traditional waterfall approach.

The Agile development model follows incremental method of software development rather than sequential, i.e., software is developed in incremental rapid cycles which results in small incremental releases. Every release is built on the functionalities of the previous release. Each release is thoroughly tested to ensure that all the bugs are detected and resolved before the product is released.

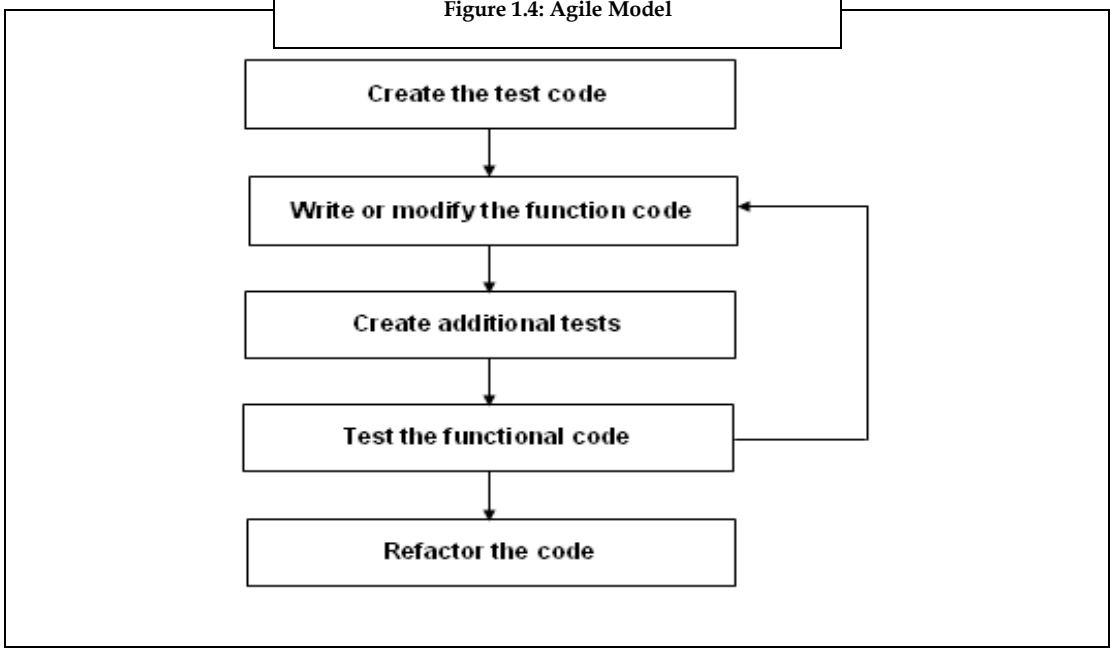
Another important aspect of the Agile model is that customers, developers, and testers constantly interact with each other during the entire development process. The tester is aware of the requirements being developed and can easily find out any gaps that exist between the developed software and requirements. This is carried out for every release that is made during the software development life cycle.

Many approaches are used to achieve agile methodology, such as Dynamic Systems Development Method (DSDM), SCRUM, and Extreme Programming (XP). Among all these approaches, Extreme Programming (XP) is the most popular and widely used approach.

In XP software development life cycle, programmers usually work in pairs. One programmer takes the responsibility of writing the code for the software and the other programmer reviews the code to ensure that it uses simple solutions and follows the best design principles and coding practices. This means the second programmer acts as a tester and tests the software as and when it is developed to find the bugs.

Test-driven development is one of the core practices of XP. It adopts feedback approach of software development where test cases are developed before the actual code is developed. The figure illustrates the various phases of test-driven development.

Figure 1.4: Agile Model



1. ***Create the Test Code:*** The developer creates the test code using an automated test framework even before the actual code is written and these test codes are submitted to the test team. The functionality of the software is developed based on the test code.
2. ***Write or Modify the Function Code:*** The functional code is written for the test codes that have cleared the test case. The actual functional code is not written until the test case requirements are met with. Once the functional code is written it is again checked using the test case. The functional code module is completed only after it clears the test cases.
3. ***Create Additional Tests:*** In this step the tester tests the module for various types of input. The tester develops various test conditions depending on the complexity of the module.
4. ***Test Functional Code:*** The functional code is tested based on the test case developed in step 3 and step 1. The steps 2 to 4 are repeated until the code clears all the test cases.
5. ***Refactor the Code:*** In this step, some changes to the code are made to make the code easy to maintain and extensible. This enables the developer to make changes to a particular module without affecting the entire application. Any new feature can be added to the existing application easily, without major modifications. It also removes any duplicate code or unused code and reduces the code complexity.

Thus, in Agile method testers have a strong role to play in development of efficient software.

Error in Intel's Pentium Microprocessor

A Mathematics professor at Lynchburg College, USA, Dr.Thomas Nicely, found a floating point division error in Intel Pentium microprocessor in the year 1994. He was computing on his Pentium-based computer to find the sum of the reciprocals of prime numbers.

He noticed that the computation result was significantly different from the theoretical values. When the same computation was carried out on another computer with a different microprocessor (486 CPU) he was able to get the correct values that obey the theoretical values. The worst and well-known case was division of 4195835 by 3145727. The correct value is 1.33382, however the Pentium's floating point unit computed and generated a value 1.33374 (Only 6 places after decimal is mentioned here), which has an error of 0.006.

Dr.Thomas Nicely reported the bug to Intel, but Intel ignored it and Nicley did not receive any proper response from Intel. Later, Nicley took the issue to public with the help of the Internet and media. Following these events, Intel publicly announced that "an error is only likely to occur [about] once in nine billion random floating points". They also mentioned in their announcement that "an average spreadsheet user could encounter this subtle flaw once in every 27,000 years of use." However, it was noted that the Intel Pentium processors output was wrong every time when such division was performed for such values.

On November 28, 1994, Intel publicly admitted the problem and issued a statement saying that it would replace pentium chips only for those who could explain the need of high accuracy in complex calculations. Industry experts, public, and media criticized this attitude of Intel. Later, in the month of December, Intel announced that it would freely replace the processor for any owner who asked for one.

The bug problem made Intel to issue an apology to its customers and the public for the way it handled the bug and issues related to it. It had to spend more than $400 million for replacing bad chips. Learning a lesson from this, Intel now reports all known problems on its official website and it also monitors customer feedbacks carefully and regularly.

Questions

1. Is Intel right in their approach to customers? How did they win customers' confidence?
2. Explain the kind of bug faced by Intel. What method would you suggest to overcome the bug?

Adapted from (http://www.willamette.edu/~mjaneba/pentprob.html)

1.4 Summary

- Software testing demonstrates that a program performs the intended functions correctly. The ultimate goal of software testing is to ensure that the product is error free.
- Detecting bugs are the most important part of a software testing process. These bugs can be an error in the program or issues that affect the quality of the software.
- Depending on the kind of error or the reason for error the bugs are classified into various types such as, bugs due to conceptual error, math bugs, logical bugs, resource bugs, co-programming bugs, and team working bugs.
- Bugs in software can occur due to various reasons such as human error, lack of communication, tight time lines, improper design logic, inefficient coding practices, and unskilled testers.
- Bugs can prove to be very costly. They not only take time to resolve affecting the project time lines and project cost, but also result in losses as the company may have to compensate the customer by

way of money or by way of replacing the defective product. It can also damage the reputation of the company.

- The Waterfall model of software development is a traditional model which follows sequential method of software development. V and Spiral models, which employ testing as an integral part of software development, are more efficient than the Waterfall model. Agile is considered to be the most advanced and efficient type of software development model. Extreme Programming (XP) is a method developed based on the agile model, which uses test driven development to develop highly efficient software.

1.5 Keywords

Confined: Within bounds or limits

IEEE: IEEE is the acronym of Institute of Electrical and Electronics Engineers. It is the world's largest professional association dedicated to advancing technological innovation and excellence for the benefit of humanity. They foster development of national and international standards.

Race Condition: A race condition occurs when a program doesn't work as it is supposed to, because of an unexpected call or ordering of events that produce contention or a clamor over the same resource.

Recursion: Recursion is a process of defining or expressing a function or procedure in terms of itself.

1.6. Self Assessment

1. State whether the following statements are true or false:
   (a) Software testing identifies the areas of weakness in an application or product.
   (b) The year 1988-2000 followed prevention oriented approach.
   (c) Testing is a process of executing a program with the intent of finding errors -- was pointed out by Myers.
   (d) Some compromises will be made in the requirements or design of the software to meet the delivery requirements.
   (e) Sometimes efficient tools such as editors, compilers, and debuggers generate wrong codes which cause errors in the software.
   (f) The analysis phase defines the software and the hardware requirements relevant for the software development process.
2. Fill in the blanks:
   (a) Software testing is an _____ and _____ process to detect a mismatch, a defect or an error.
   (b) Many bugs occur due to lack of _____ when a developer tries to modify software developed by another developer.
   (c) The _____ of fixing a bug differs depending on the development stage at which it is detected.
   (d) In Test Driven Development (Agile), the _____ is not written until the test code does not clear the test case requirements test.
3. Multiple Choice Questions
   (a) Demonstration oriented testing was followed during which of the following period?
      (i) 1957-1978 (ii) 1979-1982 (iii) 1983-1987 (iv) 1988-2000
   (b) Which of the following factors causes errors due to incorrect usage of logic or syntax?
      (i) Communication failure (ii) Human (iii) Lack of skilled testing (iv) Unrealistic timeframe

(c) Identify the task that is not performed by a software developer during software testing.

(i) Study and understand the requirements and prepare verification and validation test

(ii) Prepare test data (iii) Plan testing (iv) Automate test cases (v) Fix defects

(d) During which stage of V Model is the acceptance test plan developed?

(i) Requirements specifications (ii) Functional specifications

(iii) System design (iv) Integration test

(e) What testing is carried out after integrating the units to ensure that specifications are met?

(i) Agile testing (ii) Unit testing (iii) System testing (iv) Acceptance testing

1.7 Review Questions

1. "The general aim of testing is to affirm the quality of software systems by systematically exercising the software in carefully controlled circumstances". Justify.
2. "Deve Gelperin and William C. Hetzel classified software testing based on the goals and phases". Explain the phases.
3. What kind of bugs can you think of while testing a web page? How would you classify them?
4. Factors such as communication failure, unrealistic development timeframe, poor design logic, poor coding practices and lack of skilled testing are reasons for occurrence of bugs. Explain.
5. Bugs detected at different levels of software development life cycle have different effects on the cost incurred in resolving them. Explain.
6. Is there any difference between the water fall model and the spiral model? Discuss.
7. In V model, test cases are developed at every stage of software development. Explain.
8. Explain how you will carry out software development process using RAD model.
9. In Agile model, the customers, developers, and testers constantly interact with each other during the entire development process. Discuss.
10. Do you think that all bugs found during software testing will not be fixed? Discuss.

Answers: Self Assessment

1. (a) True (b) False (c) True (d) True (e) False (f) False
2. (a) Incremental, Iterative (b) Communication (c) Cost (d) Functional code
3. (a) 1957-1978 (b) Human (c) Fix defects (d) Requirements specifications (e) System testing

1.8 Further Readings


Books

Ron Patton, Software Testing-Second Edition, SAMS Publishing, USA
Hutcheson, Marnie L, Software Testing Fundamentals, Wiley Publishing, USA


Online link

http://www.articlesnatch.com/Article/Software-Bug-And-Their-Common-Types/594429
http://software-testing-zone.blogspot.com/2008/12/why-are-bugsdefects-in-software.html
http://tech.lds.org/index.php/newsletter-archive/238-the-cost-of-bugs
http://msdn.microsoft.com/en-us/library/ff649520.aspx
http://www.asknumbers.com/QualityAssuranceandTesting.aspx

Unit 2: Fundamentals of Software Testing

CONTENTS

Objectives

After studying this unit, you will be able to:

- Illustrate structural and functional testing strategies and techniques
- Explain static and dynamic testing
- Explain manual and automated means of testing
- Discuss the role of software tester
- State the axioms of software testing.
- List software testing terms and definition

Introduction

Testing is a vital part of any software development process. According to the IEEE definition, "Software testing is the process of analyzing a software item to detect the differences between existing and required conditions and to evaluate the features of the software item". IEEE's Guide to the Software Engineering Body of Knowledge, SWEBOK, states that "Software testing is an activity that should be done throughout the whole development process".

An efficient and reliable software product can be built with high level of planning and by following a structured development approach. Testers play a vital role in achieving high degree of efficiency in software product development. Defining the test strategy helps in obtaining a consensus of goals and objectives from the relevant stakeholders. It helps to manage expectations and to identify the types of tests that need to be conducted at all levels. Testing strategies not only help to perform efficient testing,

but also make the process of testing more organized. Choosing a suitable technique helps to define the correct methodology of approach.

The main focus of any testing technique is to make the software bug free. Therefore, it can be said that the testing techniques aim to target a specific area of the software and find bugs. Testing techniques provide methods which the tester can use to search bugs even in a highly complex and lengthy software, easily and efficiently.

Software testing depends more on what you want to test in software, and how you want to test. This will help you to decide the tools that you need to gather in order to perform the test. In deciding the nature of the testing, testers play a key role in an organization. Their contribution is vital in deciding the appropriate implementation for a given test.

2.1 Testing Strategies and Techniques

A test strategy is a concise statement that describes how the objectives of the software testing are met. The test strategy views the test event at a high level, concentrates on the objectives of the test event, the techniques that can be used and the resources that are required. It is vital that developers should plan their approach to testing at every stage of the project and establish a framework for testing the project. A testing technique can be defined as a process that ensures that the application being tested functions in a structured way. A testing strategy and technique is based on the method of testing adopted. Let us now learn the different methods of testing.

The three main methods of testing are the following:

1. Structural or functional testing
2. Static or dynamic testing
3. Manual or automated testing

2.1.1 Structural versus Functional Testing

If the test cases are developed to check the actual structure of the program code, then it is called structural testing. Structural testing is also known as white box testing, where the tester checks the actual code of the software. However, in functional testing, the tester checks only the behavior of the software and will not check the actual code. The tester only checks the response of the software for predefined inputs and tests whether the software produces the desired output. Therefore, this is called black box testing.

If the test cases are developed to check how the entire system works, then it is called functional testing. During functional testing, the tester uses test cases to check how the software works, i.e., whether it produces the desired outputs for a set of given inputs.

Example: Functional testing for a calculator could check whether the software does the addition operation correctly.

We will cover Structural and Functional Testing in detail in unit 3.

2.1.2 Static versus Dynamic Testing

Static testing refers to the analysis of the program, which is carried out without executing the program. This is a typical white box testing technique, where the developer checks the code to find errors in it. It is preventive in nature and is completed in the verification phase. The common methods include feasibility review and code review.

Example: Software developers perform syntax check to test syntactical correctness.

Dynamic testing refers to the analysis of the program in its executable form. This is performed by supplying valid entries and is validated against the expected results. It is a curative method and is performed during the validation phase.

Example: Software developers perform unit test to check for correctness in a module.

Thus, static testing is done to check mainly the correctness and logic of the code whereas, dynamic testing is performed to check the response of the system for predefined inputs.


Notes

Some of the important differences between static testing that makes it more effective and efficient are:

1. Since static testing is carried out during the initial stage, it is cost effective compared to dynamic testing, which is carried out once the entire software, a module, or unit is complete.
2. Static testing detects bugs at the earliest and hence the time required to fix them is less.

The process of static test is very tedious since every line of the software has to be checked by the developers. However, many tools have been developed to address this issue. These tools enable the developers to perform the static test faster.

In order to make the software bug free, it is very important to carry out both static and dynamic testing.


Did you know? There is a popular myth that the goal of software testing is 100 percent defect-free code. In reality, complex applications will never be free of defects. Software testing can detect about 90% of errors, but the rest are generally found only when the system goes live.

2.1.3 Manual versus Automated Testing

When the software is tested by people to find the bugs, it is called manual testing process. During this test, the tester acts as an end user and uses all the features of the software, and checks to ensure that they behave correctly.


Example: Performing a manual test for a login screen involves some of the following:

1. Checking whether username and password can be entered.
2. Implementing masking of password character.
3. Verifying whether the screen navigates to next page if valid login details are provided.
4. Checking the error message displayed when incorrect login details are entered.
5. Checking the maximum number of characters that can be entered in the login and password fields.

In automated testing, a software program, commonly referred to as 'a testing tool', runs the test software, provides proper inputs, and checks the output against the expected output. A tester writes the test case, and the automated testing tools run the test software according to the test case without any human intervention. At the end of the test, a detailed report is generated to indicate the result of any condition which is defined in the test case.

Manual testing requires human intervention at every stage of the testing process right from writing test cases, providing the inputs, recording the output and analyzing the actual and expected output. There is always a chance for error in manual testing. It also requires more time to perform the test.

Automated testing uses testing tools to perform the test. These tools need initial human intervention for supplying test cases. These tools are very powerful and reduce the time required for testing and are very efficient in finding out bugs in software. However, huge investment is required for using automated testing tools, since the tool has to be purchased from the vendor and some investment is also required for training.


Example: HP's Quality center is a popular automation tool used for testing and quality assurance.

Limitations of Manual Testing:

1. During the process of testing, the possibility of recurrence of a bug can actually have an impact on the time taken for testing the software.
2. Frequent changes in the user scenarios can lead to high maintenance costs in manual testing.
3. Repetitive tests reduce the cycle time for executing a test, and hence automation testing is opted for.


Notes

Depending on the limitations of time and cost factor, automation tests are preferred over manual tests.

Advantages of Automation Testing

1. It reduces the time consumed to perform repetitive tests.
2. It requires less human effort and less number of resources.
3. It generates a test report which provides information of the test execution unlike a manual test which is written and documented.
4. It helps in regression testing (Testing which is done to check whether the changes made in a module affects the working of the other existing modules) and also helps in re-running tests against new releases.
5. It helps in testing large sequences of data and transactions and also in randomly searching for errors in the software.
6. It helps in testing several simultaneous users at a time virtually and can also analyze the load generated for the program -- which cannot be done in manual testing.
7. It helps in testing web-based systems for performance reliability.

2.2 Role of a Software Tester

The role of a software tester is pivotal in the testing life cycle, as he/she is responsible for the activities carried out in a testing life cycle. The main goal of a software tester is to find bugs, find them early and also ensure that the changes incorporated due to the correction does not affect other functionalities.

Software developers have good problem solving skills and always demonstrate that the software works as intended. A tester, on the other hand, demonstrates the weakness of an application. The application developed is checked with test cases or configurations, which gives errors or unexpected results to show where exactly the software breaks.

Some of the responsibilities of a software tester during the process of testing are:

1. Understanding the product/application by analyzing the specifications.
2. Implementing a test strategy, which includes writing appropriate test plans, prioritizing the testing by assessing the risks, and setting up test data.
3. Setting up the test environment for manual and automated tests.
4. Providing reports that list the product defects and metrics.


Caution

There is a serious misconception that software testing is only an entry- level position in the software industry and is a low pay job. There are various profiles that testing offers which includes the positions of software test technicians, test engineers, test leads, test managers, quality assurance engineers, quality auditors, and quality managers. According to a survey across 367 IT organizations in 22 countries, the global software testing market is estimated to reach US$56 billion by the year 2013 and requires 30,000 professionals globally every year.

2.2.1 Tasks of a Software Tester

Software testers are quality champions who are involved in various activities of the testing life cycle. Testers co-ordinate with developers and conduct test case reviews of project areas.

The tasks of a software tester are as follows:

1. To ensure that the test methodology, techniques, and standards are established, developed and documented.
2. To study and understand the requirements and accordingly prepare verification and validation test plans.
3. To impact product quality by understanding customer needs.
4. To develop test plans, test scenarios, and test cases.
5. To prepare the test data and execute test cases.
6. To automate test cases.
7. To perform both verification and validation testing of the hardware and the software.
8. To prepare test reports and maintain test records.
9. To submit reports that details the schedule progress, the defects and the usability of the product.
10. To track defects and ensure closure of defects through reviews.


Notes

Goal of a software tester
The main goal of a software tester is to find bugs as early as possible and ensure that they are fixed at the earliest.

2.2.2 Qualities of a Software Tester

Testing profession requires a methodical and disciplined approach and hence a good software tester should possess strong analytical skills with good domain knowledge.

The following are the qualities of a good software tester:

1. Have a strong desire for quality.
2. Be explorative in approach to venture unknown situations.
3. Have a creative and relentless approach to discover bugs.
4. Be tactful and diplomatic with developers while conveying where the software lacks.
5. Possess good ability to understand customer needs.
6. Be able to compromise between the available resources and be in a position to focus on the most likely areas of bugs when there is insufficient time.
7. Possess good judgment skills to assess high-risk areas of an application.
8. Be sharp enough to observe the small changes.
9. Have a good understanding about the software development process.
10. Be technically aware of testing methods, tools, and criteria.

2.3 Software Testing Axioms

The world of software testing is not just dictated by the models discussed and followed. In reality there are many trade-offs software testing effort faces. In the current day scenario, it is less likely that clients are able to determine every requirement analysis aspect in one-go.

Requirements can keep changing during the course of time. Hence there are more chances for some of the following realities to occur.

(a) The specification might not correspond to the customer's needs perfectly.

(b) Many a time, the time available for testing would not be comprehensive to cover all aspects of testing.

(c) Tradeoffs and concessions are inevitable.

Realities of Software Testing

A detailed specification which can perfectly meet the needs of a customer is not given and there is insufficient time to test the software in its entirety. However, if one aims to become a good software tester, he or she needs to:

(a) Identify the ideal process involved.

(b) Identify the bugs and problems and realize how they affect the project.

Let us now familiarize with a few axioms that are facts in the life of a software tester.

Axiom 1

It is impossible to test a program completely.

A tester may not be completely sure about the number of test cases needed to exhaustively test an application, for example

Example: Testing an MS Word document with all possible test cases covering all functions would be a difficult task to complete.

The only way to absolutely ensure that the software works perfectly is to test it with all possible inputs and observe and monitor its outputs. At times, questions do arise about the number of possible inputs being very large, the number of possible outputs being very large, the number of paths through the software being very large or the software specification itself being open to interpretation.

Axiom 2

Software testing is a risk-based exercise.

When one does not test the software with all the possible inputs, they may end up taking a fair amount of risk, wherein there are possibilities of skipping some of the inputs which work correctly. At this stage, one faces the risk of skipping inputs which can cause a failure and may lead to financial loss or loss of security or even loss of life. This brings a tremendous amount of pressure on a software tester.

Software testing is considered to be a risk-based regime of practice, where one can find that:

(a) Testing too much can result in high developmental costs.

(b) Testing too little can result in the failure of the developed software, which can incur heavy cost to an organization.

(c) The general cost involved in testing the number of missed bugs, over testing and under testing are more.

Axiom 3

Testing cannot show the absence of bugs.

It is not an easy task to fully ascertain that the software is totally bug free. During the course of testing, we cannot completely eliminate the bugs, unless the software is dismantled to the foundation stage. Software testing is a process which can reveal the existence of a bug, but cannot reveal that there are no bugs in the software. Although, tests are performed to report and fix the bugs, it is not possible to guarantee bug free software.

Axiom 4

The more bugs you find, the more bugs there are.

Bugs in real life and bugs in software are very much alike. They come in groups and when you happen to accidentally notice one of them, there are possibilities of finding another one very soon. Most often a tester finds a bug only after long hours of testing, and when he/she encounters one of the bugs, he/she would soon find another one too.

The reason for finding bugs at frequent intervals could be due to programmer's errors, where mistakes often are repeated or different programmers handling a module may have different habits of coding. Bugs are considered to be the tip of the ice berg but can cause huge problems to architecture of the software. However, the inverse of the same is also very true. If you do not find a bug, it is clear that there are indeed no bugs and that the software has indeed been written well.

Axiom 5

Not all bugs found will be fixed.

The reality about software testing is that, in spite of all the effort put in to find a bug, it might not be fixed. Hence, it is required that software testers to have good judgmental skills and make trade-offs, risk-based decisions for every bug they find.

The reasons why all bugs cannot be fixed are:

(a) ***There's Not Enough Time:*** Every project has several software features where only few people are involved in coding and testing and hence it becomes difficult to adhere to stringent time schedules to complete assigned tasks.

(b) ***It's Really Not a Bug:*** A bug found need not be a bug but could turn out to be its characteristic feature. There are possibilities of mistaking features as bugs.

(c) ***It's Too Risky to Fix:*** Most often, it has been found that it is indeed very risky to fix bugs. One should first fix the bug which might cause other bugs to appear and it is also necessary to ensure that last moment changes of software do not occur during the release of a product.

(d) ***It's Just Not Worth it:*** Some of the bugs which affect the fringe features are dismissed.

Axiom 6

It is difficult to say when a bug is indeed a bug.

We need to analyze the following:

(a) When a problem in the software is not discovered - is it a bug?

(b) Is it necessary for a bug to be observable?

Did you Know? The bugs which remain undiscovered and exist in a system for over a period of time may exist for more than one version and there are also possibilities of the bug being identified after the release. This is known as latent bug. A latent bug does not cause damage for some time, however; they reveal themselves at a later point of time.

Example: The Y2K problem is the best example of a latent bug. Initially the beginning of the year was allocated with only two numeric fields, when it actually required four numeric fields. This problem existed as a latent bug in the system without causing any damage; however the problem was identified and fixed before the year 2000.

Axiom 7

Specifications are never final.

The changing specifications make it difficult for complete testing to take place. Specifications can change due to:

(a) Fierce Competition.

(b) Rapid release cycles.

(c) Change requirements.

Axiom 8

Software testers are not the most popular members of a project.

Software testers do have goals to:

(a) Find bugs early and ensure that they are fixed as early as possible.

(b) Ensure that they adhere to professional behavior without losing their temper.

Axiom 9

Software testing is a disciplined and technical profession.

Initially software testers were untrained and did not follow any methodology, as the software was simpler and manageable. However, testing has now become a matured discipline and supports sophisticated techniques with good support of tools and also provides a rewarding career for the testers.

2.4 Software Testing Terms and Definitions

Let us understand some important software testing terms:

Software Quality:

Software quality is impacted by bugs. It is essential that the software is bug free or defect free and meets the requirements, specifications, and expectations of the client.

Verification and Validation:

Verification is the process of discovering the possible failures in the software before the commencement of the testing phase. It involves reviews, inspections, meetings, code reviews, and specifications. It answers the question." Are we building the product right?"

Validation occurs after the verification process and the actual testing of the product happens at a later stage. Defects which occur due to discrepancies in functionality and specifications are detected in this phase. It answers the question, "Are we building the right product?"

Quality Assurance (QA) Vs Quality Control:

Quality Assurance and Quality Control are terms that define the quality management activities of a project. While quality assurance refers to the planned and systematic activities that monitor and ensure that the development and maintenance process meets its objectives, quality control refers to a set of activities that are designed to evaluate a developed product.

Quality assurance is more a verification process, whereas quality control is more a validation process. The activities are more generic and can encompass the whole development process. The activities of quality assurance can be performed while the product is being developed, whereas the activities of quality control are performed after the product is being developed.

The main aim of quality assurance is to prevent defects, and thus it focuses on the process of product or application building. Quality assurance can be performed by a manager or even a third party professional. Quality assurance ensures that the process is well defined and is performed through the life cycle of the product. The activities include quality management review functions like process checklists, project audits, and standards development for development in coding. Quality assurance

identifies areas of improvement in the processes through prevention plans and also ensures that the processes followed are effective.

The main aim of quality control is to evaluate whether the deliverables are of acceptable quality. The focus thus is to detect defects and correction of these defects (correction oriented). Normally, the activities of quality control are performed by the testing team of an organization. The activities of quality control ensure data integrity, correctness and completeness. The various quality control activities include inspections, reviews, and walk throughs of design, code and documentation. Quality control addresses bugs, errors and omissions.

In simple terms, the principles by which quality assurance works is to check whether the product is "fit for purpose" and also to ensure that it is built "right the first time". Quality control works on the principle, "fix the problem". In software quality control, testing methods like unit testing, integration testing and system testing are the commonly used methods.

Test Plan:

A test plan is a document which gives information about the objectives, scope, approach and the various attributes that the testing project must focus on.

Test Case:

A test case is a document and is the smallest unit of testing. It has a developed set of inputs, execution preconditions and expected outcomes for a specific objective. This is done to ascertain that the feature of a particular application is working as specified.

A test case generally contains test case identifier, test case name, objective, test conditions/setup, input data requirements, steps, and expected results.

Table 2.1 below describes common terms and definitions associated with software testing:

Table 2.1 Terms and Definitions

Terms	Definitions
Acceptance Testing	Acceptance testing is conducted by the customer or the user to check whether the software product meets the requirements.
Agile Testing	Agile testing is for testing the software product from the customer perspective at an early stage. Testing is carried out once the codes become available.
Automated Testing	Automated testing is a procedure of using automated tools to execute tests.
Bug	A bug is an error or a defect in a program which is unintended.
Debugging	Debugging is the process of detecting and eliminating the causes of software errors.
Defect	A defect is an error or non-conformance of a specific program.
Integration Testing	Integration testing is performed on interfaces between components.
System Testing	System testing relates to testing the system after integrating the units, to ensure that specifications are met.
Unit Testing	Unit testing is the process of testing the basic unit of software, which is the smallest testable piece of software.

2.5 Summary

- Testing strategies and techniques help the tester to carry out the test efficiently to find maximum number of bugs in software.
- A static test is carried out to check for bugs in software before the software is compiled and run. A dynamic test is carried out after the software is compiled and executed.
- Manual test involves performing the test without taking the support of any testing tools. All the test activities such as writing the test case, providing the inputs, recording the output and comparing the expected and obtained output are carried out by the testers manually.
- Automated testing is completely dependent on software testing tools. The tester writes the test cases and the tools perform the test activities.
- A good software tester must be creative, explorative and should be able to identify the ideal process involved for testing and realize the impact of bugs on a project.
- Structural testing techniques check the occurrence of bugs in the test software using the actual codes of the software. The tester works with the source code of the software while performing the test.
- In a functional test, the tester is not aware of the actual working of the software. The test analysis is performed based on the outputs that the software generates for various inputs. The bugs are detected by comparing the expected output with the obtained output.
- Software axioms are self-evident facts which bring to light the real-life situations faced by testers.

2.6 Keywords

Axioms: Axioms are postulates which are accepted on their own merits without any mathematical rule. They require no proof. They are formulas that are not derived from others but are self-evident facts.

SWEBOK: Software Engineering Body Of knowledge is a product of the Software Engineering Coordinating Committee which is sponsored by the IEEE Computer Society. They define knowledge areas within software engineering. This includes Software requirements, Software design, Software construction, Software testing, and Software maintenance.

Test Strategy: Test Strategy is an outline that describes the testing aspects to the respective stakeholders. The Stake holders could be project managers, testers, and developers. Test strategy discusses the team's approach to the testing process.

Testing Tool: Automation of tests is carried out with the help of a testing tool. The tool is software which controls the execution of tests. The software compares the actual and predicted outcomes, the setting up of test conditions and the test control and test reporting functions.

2.7 Self Assessment

1. State whether the following statements are true or false:
   (a) The test strategy views the test event at high level, concentrates on the objectives of the test event, the techniques that can be used and the resources that are required.
   (b) During the process of automation testing, the possibility of recurrence of a bug several times can actually have an impact on the time taken for testing the software.
   (c) In white box testing, as the tester has the knowledge of internal coding, it is very easy to develop test cases to test the software effectively.
   (d) While performing a software test, the tester should first begin the test with test to fail and check whether the software works fine without any bugs.
   (e) The tester enters erratic or irrelevant data and checks the response of the software while performing mutation testing.

2. Fill in the blanks:

   (a) During functional testing, the tester checks only the _____ of the software and will not check the actual code.

   (b) Static testing is performed to check the bugs in the software using the _____ of the respective software.

   (c) _____ testing is also called as glass box testing.

   (d) Developing efficient_____ are very essential during testing.

3. Multiple Choice Questions

   (a) Identify the testing technique that is used to test how the actual code works.

      (i) Structural testing

      (ii) Functional testing

      (iii) Static testing

      (iv) Black box testing

   (b) What testing is performed to check the response of the system for predefined inputs?

      (i) Static testing

      (ii) Dynamic structural testing

      (iii) Dynamic testing

      (iv) Structural testing

   (c) Which of the following would be an optional skill of a tester?

      (i) Be explorative

      (ii) Be sharp

      (iii) Be tactful and diplomatic

      (iv) Be knowledgeable in programming language

Answers: Self Assessment

1. (a) True (b) False (c) True

   (d) False (e) False

2. (a) Behavior (b) Source Code (c)White box

   (d) Test cases

3. (a) Structural testing (b) Dynamic testing (c) Be knowledgeable in programming language

2.8 Review Questions

1. How does test strategy differ from the test technique? Substantiate how planning a strategy helps in efficient testing.
2. Suppose you are recruited as a tester in a software company, what qualities are you expected to exhibit?
3. Do you think software testers need to be just knowledgeable in their domain? Are there any specific soft skills they are expected to exhibit?
4. "The process of static test is very tedious". Explain why.
5. Is there any difference between Verification and Validation? Discuss
6. "Not all bugs found will be fixed. " How would you substantiate this axiom?

7. "Automated testing uses testing tools to perform the test." Does this mean that there is no human intervention needed for automated testing?
8. Is quality assurance and quality control mutually exclusive quality initiatives? Which activity is closer to testing?
9. "Software testing is a risk-based exercise." Explain.

2.9 Further Readings


Books

Ron Patton, Software Testing-Second Edition, SAMS Publishing, USA
Hutcheson, Marnie L, Software Testing Fundamentals, Wiley Publishing, USA
Kassem A. Saleh, Software Engineering, J.Ross Publishing, 2009, US


Online link

http://qastation.wordpress.com/2008/04/21/static-testing-vs-dynamic-testing/
http://www.adager.com/vesoft/automatedtesting.html
http://www.scribd.com/doc/2453259/Testing-Techniques-and-Strategies
http://www.testinggeek.com/index.php/testing-articles/137-equivalence-partitioning-introduction
http://www.cc.gatech.edu/classes/cs3302_98_summer/7-02-unittest/sld009.htm
http://www.slideshare.net/nworah/types-of-software-testing
http://www.ece.cmu.edu/~koopman/des_s99/sw_testing/
http://www.softwaretestinghelp.com/what-is-boundary-value-analysis-and-equivalence-partitioning/

Unit 3: Black Box Testing

CONTENTS

Objectives

After studying this unit, you will be able to:

- Illustrate structural and functional testing strategies and techniques
- Explain static black box testing techniques
- Explain dynamic black box testing techniques
- Discuss test to pass and test to fail
- Explain equivalence partitioning, data testing, and state testing
- Explain random testing and mutation testing

Introduction

We are aware that the Testing Technique specifies a strategy that is used in testing select input test cases and analyze test results. There are various testing aspects that are revealed through the Structural and Functional Testing. When the features and operational behavior of the product needs to be tested, Functional Testing or Black Box Testing can be approached. The advantage of this kind of testing is that they totally ignore the internal workings of the system.

Organizations have to make the right choice between Structural and Functional testing. With increasingly complex applications, Total Cost of Ownership (TCO) and Return on Investment (ROI) are two criteria that favor the Black Box testing technique. However, if the strength of the application needs to be introspected, or if the application has to be checked for stability or needs to be ascertained for thoroughness, it would have to undergo white box testing.

3.1 Structural and Functional Testing

Structural and functional testing are two important types of software testing. Structural and functional testing are also called as white box and black box testing.

3.1.1 Black Box Testing

Black box testing, also termed as behavioral testing, checks if the software works as per the desired requirements or specifications. It is called black box testing because the tester performs the tests without knowing the internal logic of how exactly the software works. He/she focuses on the outputs generated in response to selected inputs and execution conditions.

Did you know? In neural networking or Artificial Intelligence simulation, a black box is used to describe the constantly changing section of the program environment which a programmer cannot test easily.

Developing efficient test cases is very essential during black box testing. Since the tester has no knowledge of the internal working of the software, they need to rely completely on the analysis of the transformation of the inputs to the outputs based on which they find bugs in the software. This test enables the tester to know whether or not the software does what it is supposed to do. The functional specifications or requirements of the software provide the information about the software functionalities.

Example: Testing search engine is a good example for black box testing. You are not aware of the processes that work behind the search engine to provide the desired information. While testing a search engine you provide input in the form of words or characters, and check for output parameters such as relevance of the search result, time taken to perform the search or the order of listing the search result.

Advantages of Black Box Testing

Black box testing has many advantages, which include the following:

1. Testers do not have to understand the internal working of the software, and it is easy to create test cases with the perspective of an end user.
2. The testers mainly deal with the Graphic User Interfaces (GUI) for output, and they do not spend time analyzing the internal interfaces. Therefore, test cases can be developed quickly and easily.
3. As soon as the specification of the product is complete, the test cases can be designed.
4. Black box testing helps to expose any ambiguities or inconsistencies in the specifications, and tests are carried out from a user's perspective.

Example: Testing the functions of an ATM is a good example of black box testing. The tester acts as a customer who is using the ATM and checks the functions of the machine. He/She does not know the internal working of the logic. The test cases are developed to check the functions through the GUI of the ATM such as change in display of the GUI when card is detected, masking the password or navigating from main menu to a specific function.

Disadvantages of Black Box Testing

1. A tester can test only a small number of possible inputs and it is highly impossible to test every possible input stream.
2. It is very difficult to design test cases if specifications are not clear and concise.
3. Situations, such as unnecessary repetition of test inputs, can occur if the tester is not informed of test cases that the programmer has already tested.

4. This type of testing cannot be focused on specific segments of function that may be very complex, therefore bugs can go undetected.

Example: When there is a complex system to be tested like the online Indian railways booking system, it is difficult to identify tricky inputs and write test cases to cover all possible scenarios. .

Performing a black box test, the tester attempts to find the following errors based on the behavior of the software:

Missing or incorrect functionality.

1. Errors in interface.
2. Data structure errors.
3. Performance errors.
4. Initialization and terminal errors.

As a part of black box testing strategy, it is very important to use test monitoring tools. This is needed to track the tests that have already been executed, to avoid repetition and to aid in the software maintenance.

3.1.2 White Box Testing

White box testing is also called as glass box testing. In this test, the tester focuses on the structure of the software code. The tester develops test cases to check the logical working of the software code.

Black box testing helps to answer the validation question "are we building the right software?", but white box testing helps to answer the verification question "are we building the software right?"

Example: White box testing helps the tester to check how exactly the calculator performs the addition of two numbers and if it is the efficient way to perform the addition operation.

In white box testing, each software module is tested independently. The tester has to develop test cases not only to test the individual module of the software, but also to test how exactly the modules interact with each other when software is executed. All the tests are carried out at the source code level. The tester checks all the parameters of the code such as efficiency of the code written, branching statements, internal logic of the module, interfaces between external hardware and internal module, memory organization, code clarity, and so on. Therefore, the test cases must be carefully designed in order to cover the internal working of the application.

The tester who writes the test cases to perform white box testing has to be very well aware of the language and logic used to develop the test software. He/she needs to know programming concepts as well.

Advantages of White Box Testing

1. As the tester has the knowledge of internal coding, it is very easy to develop test cases to test the software effectively.
2. Testing is carried out at the code level; hence it helps in optimizing the code.
3. Unnecessary or extra lines of code which can generate hidden bugs can be removed.

Example: A test case to check for bugs in the loops that are used in a software application, should include the following situations:

1. If the loop iterates zero times.
2. If the loop iterates once.
3. If the loop iterates twice.
4. If the loop iterates several times.
5. If the loop iterates n - 1 times.
6. If the loop iterates n times.
7. If the loop iterates n + 1 times.
8. If the loop iterates infinite times.

Disadvantage of White Box Testing

1. It is highly impossible to check every code to find out the hidden errors or bugs, which may cause problems that lead to failure of the software.
2. Skilled testers are required to carry out this test, which increases the cost.
3. The time required to carry out this test for complex software is very high.

Both black and white box testing has its pros and cons. It is very important to understand the need for the kind of testing before selecting any of them. Experts believe that, if black and white testing is carried out together, it would yield better results. Therefore, the testing team must strike a balance depending on the project requirement to adopt both black and white box testing to test the software.

Notes

Selecting the right testing method for testing the software is very important, since both black and white box testing methodologies have their merits. Following are a few questions which can help you in taking the right approach:

1. Who will be the users of the application?
2. Prior to release, which parts of the application must be tested and why?
3. When do we make significant changes to the User Interfaces and will this affect the actual code of the application?
4. Where is the application likely to be installed?
5. How will end users be using the application?
6. Which platforms does the application need to support after installation?

3.2 Static Black Box Testing and Dynamic Black Box Testing Techniques

Black box testing techniques can be broadly classified into two types, static and dynamic black box testing.

Static Black Box Testing Techniques

Static testing, as the name suggests, is used to test the software without compilation. When the test is performed to check the specifications, it is called static black box testing. Usually the specification is a document that provides information of the software functionalities. This document is created during the initial stages of the Software Development Life Cycle (SDLC) based on the input from the customer and designer. The tester carefully performs static black box testing and checks for bugs. The focus of static

black box testing is to check for completeness or appropriateness of the product or application developed.

The specification document not only lists the functionalities of the software, but it also provides vital information of the software to the user or customer. The user or customer depends completely on the specification document to know the software. Therefore, it is very important that this document is free from bugs.

The test is more often a research, since the tester should make sure that no vital information is missed or incorrect information is provided in the document. Another important aspect of this test is to find confusing and misinterpreted information in the software application. Any such information is considered as a bug. The tester must understand the customer's or end user's expectations and make sure that the document meets these requirements.

Example: The mobile user manual is an example of specification document. The user needs this manual to operate the mobile applications and to know all the features that are available in the mobile. As a tester when you test the mobile as per manual, you need to test it with an end user perspective and make sure that the manual meets all the quality requirements. The specification should be correct, clear, and complete with all the information of features available in the mobile model.

Static black box testing - High Level Specification Test Technique

Testing the specification of a document is considered static, since we do not execute a program. The first step in testing the specification through static black box testing is not to spot on the errors and bugs. Rather, it requires a methodical approach to view the specification from a high level. The specification should be reviewed for the fundamental explanation. The specification should be checked if it is complete and if there has been any omissions. Static black box testing is more research oriented and the research helps to understand how the specification is organized and the reason behind the organization of the specification. The first step in presenting the same is to view the specification from the perspective of the customer who would be using the software. It is important to understand and meet customer expectations.

An important criterion while performing static black box testing is that of following standards and guidelines. Standards have to be mandatorily followed whereas following guidelines are subjective to the requirements of the product. Every company follows their own standards and guidelines (along with international and national standards and guidelines) to develop specification documents. Tests should make sure that the document strictly follows these standards and guidelines. Any violation of these will be treated as a bug and has to be corrected. This includes the page size, color patterns, style, font size, and so on.

As a tester, you have to research on what should appear on the software. Standards and guidelines are formed based on certain rules followed by software developed. Let us now examine some examples of the various sources from where standards and guidelines can be adapted:

1. ***Conventions Followed by Corporates:*** The software should adhere to the terms and conventions used by the company.
2. ***Industry Requirements:*** Certain Industry segments like medical, pharmaceutical, and financial industries follow their own conventions while developing software.
3. ***Government Standards:*** Government agencies follow rules stipulated by them. For example, Military standards are unique.
4. ***Graphical User Interface (GUI):*** Software that works under Microsoft Windows or Macintosh has separate published standards and guidelines that dictate the look and feel of the user.
5. ***Security Standards:*** Any software developed has to meet certain security standards or levels. In some cases, they need to be certified that they meet the necessary criteria.


Notes

Different standards and guidelines are followed for e-manuals and printed manuals. The tester has to carefully examine them during the testing process.

The software developed can also be benchmarked against products developed by competitors. Some of the areas of assessment include its complexity, features, quality and security.


Example: Most of the technical documents are based on Microsoft Manual of Style for Technical Publication (MSTP) which defines standards and guidelines for publishing technical documents such as user manuals, installation guide and operating manual.

Static Black Box Testing - Low level Specification Test Technique

High level specification testing focuses on external influences, whereas low level specification test focuses on the following eight attributes:

1. ***Completeness:*** The specification should bring complete information of the product
2. ***Accuracy:*** The defined goals of the proposed specification should be addressed without any errors.
3. ***Precision, Unambiguity and Clarity:*** The description should be easy to comprehend. The content has to be exact and not vague.
4. ***Relevance:*** Explanations are necessary and should be traceable to the requirements of the customer.
5. ***Feasibility:*** The feature must be implemented with the available personnel, tools and resources, with no additional cost.
6. ***Code-free Explanation:*** The specification should define the product and should avoid unnecessary explanations of the technology, design and architecture.
7. ***Testable:*** The feature explained should be testable.

Since Low-level specification looks out for clarity of the explanation, a tester has to keenly test whenever he/she encounters the following words:

1. ***Always, None, All, Never:*** The above mentioned cases denote a certain case. A tester has to foresee where the conditions could be violated
2. ***Certainly, Clearly, Obviously, Evidently:*** These words are persuasive words which should be tested.
3. ***Etc., And So on:*** Specifications need to be absolute or should be explained with no confusion. It is best that such words are avoided.
4. ***Some, Sometimes, Often, Usually, Mostly:*** Vague words like these need to be avoided.
5. ***Good, Fast, Cheap, Efficient, Small:*** Unquantifiable terms such as these need to be avoided.

The above is a brief list of how low-level specification is carried out. The intention of the low-level technique is to assure that all details are defined correctly.

Dynamic Black Box Testing

Static black box testing is carried out without the tester executing the code, whereas dynamic black box testing is carried out with data. It is termed "dynamic" since the tester is able to observe the changes exhibited by the system. The test is carried out by providing pre-defined inputs and the outputs are recorded. These outputs are compared with the correct output, and the variation that exists between the actual output and desired output are segregated as bugs. These tests are carried out using test cases.

The test cases have to be defined effectively in order to find the bugs. The entire test process will depend on the test cases.

Example: A software application undergoes a dynamic black box test. The test case defines that the application has to produce an output D when the inputs A and B are given. If the application gives an output C, then the application fails the test case and this is a bug.

There are various techniques used to perform dynamic black box testing. Some of the important techniques are test-to-pass and test-to-fail, equivalence partitioning, data testing and state testing. We will be discussing them in detail in subsequent sections.

3.2.1 Test to Pass and Test to Fail

The test-to-pass approach checks for the standard functions of the program. However, the test cases focus mainly to check the normal operation of the software. The test cases do not push the software to its limit or will not try to break the software. They try to find out the bugs by operating the software under normal conditions.

The test-to-fail approach test cases push the software to its extreme. The main focus is to push the software to its limit and check for bugs when the software is operated under extreme conditions. The tester tries to provide extreme and erroneous values as inputs to check how the software can be broken.

Example: A calculator application is developed to calculate the average of more than 1000 integer values. The calculator first has to be set to "Find average" mode and all the values whose average has to be found will be entered. Once all the values have been entered the “equal” button can be clicked to display the average of all the entered numbers. As a tester when you perform test-to-pass, you calculate the average for few hundred values, but will not cross the thousand values mark. This is testing the application by providing the normal input.

When you perform test-to-fail, you provide more than thousand values and check the output of the software. You even take it further by providing more than two thousand values and check the output produced by the application. Such inputs might overload the application. This helps the tester to check the maximum number of values for which the application can find the average.

It is important to note that while performing the test, the tester should first begin the test with test-to-pass, and check whether the software works fine without any bugs. Once it clears test-to-pass, the tester can perform test-to-fail.

3.2.2 Equivalence Partitioning

Equivalence partitioning, also called as equivalence classing, is a process of classifying the test cases and grouping them into different categories or classes. This method helps to reduce the number of test cases to a finite number of test cases without compromising on the quality of the test being carried out.

The main objective of partitioning is to identify the test cases that perform same kind of testing and generate similar output. Since the test cases within one class are considered to be equivalent, picking up one test case from each class would be sufficient to detect the bugs in the software. This technique helps to reduce the volume of test cases considerably.

The class can be divided into two types, valid class and invalid class. All the classes which fall under valid type are values that satisfy the condition. Classes that do not satisfy the condition fall under the invalid class.

Example: A software module checks the age of the individual who has applied for a driving license. Any individual who is above 18 years and less than or equal to 60 years is eligible to apply for the driving license.

Condition: Eligibility criteria for driving license
Valid class: Age between 18 to 60 years
Invalid class: 1. Age below 18 years
2. Age above 60 years

Here, we have three classes -- age 'between 18 to 60', 'below 18', and 'above 60' years. Say, you have a test case where you want to check what the output of the software is when you enter the age as 45. This test case falls under the case "Age between 18 to 60 years". The test cases that try to find the response of the software for the ages that range from 18 to 60 will fall under this class. Similarly, the test cases that provide the age which is less than 18 fall under the class "Age below 18 years", and test cases that provide the age which is greater than 60 fall under the class "Age above 60 years".

Notes

Some of the guidelines that are used to define Equivalence classes are:

1. If the test case input condition is a range, then you can define a minimum of one valid and two invalid equivalence classes.
2. If the test case input condition is a specific value, then you can define one valid and one invalid equivalence class.
3. If the test case input condition is a member of a set, then you can define one valid and one invalid equivalence class.
4. If the input condition is a Boolean value, then you can define one valid and one invalid equivalence class.

3.2.3 Data Testing

Any software can be divided into two parts, *Data* which consists of inputs from keyboard, mouse or disk files, and *Program* which consists of code of the program which specifies the logic or syntax.

Data testing refers to testing the data, which includes the data that the user inputs to the software and the output he/she receives. Even the intermediate values that are generated, but are not displayed on the output device are also vital data when it comes to debugging the software. Based on the concept of equivalence partitioning, the testing can be carried out on the following four levels.

1. Boundary conditions
2. Sub-boundary conditions
3. Nulls
4. Bad data

Boundary Conditions

Software certainly works fine under normal conditions, but it is important to test whether the software can operate properly under extreme conditions. Every application has a limit or maximum and minimum values it can process, and the tester has to identify these extreme limits to prepare test cases. These extreme values are provided as inputs, and tested for occurrence of bugs. Depending on the kind of application, the boundary conditions vary.

Testing the boundary is done by adding one, or a bit more, to the maximum value. It could follow the rule like the following:

1. First value -1 and/or Last value +1
2. Start conditions -1 and/or Finish condition +1
3. Less than Empty condition and/ or More than Full condition
4. Minimum value -1 and/ or Maximum value +1

Example:

1. A text field on a web page allows the user to enter up to 255 characters. The boundary values can be between 1 and 255 characters. The tester can enter only one character or 255 characters in the text field. 254 or 256 characters can be considered as boundary values to test the text field of the web page.
2. CD writer software can write 256k bytes of data on to a CD. The boundary value to test this software can be 1k bytes or 256 k bytes. The tester can provide the software 1 k bytes or 256k bytes of data to write it on the CD.

Sub-Boundary Conditions

Normal boundary conditions are based on the values that the user enters, but sub-boundary conditions are system-specific. These are the values that are related to the system hardware on which the software runs. Even though these values are not relevant to the end user, the tester has to test them, since it will result in unidentified bugs.

Powers of two is a very good example for sub-boundary condition.

Example: The decimal range of a byte is 0 to 255, which can be represented in binary as 0000 to 1111. The tester can write test cases to make the software handle these values. Pass the sub-boundary value 0, 1, 255 or 256 and check how the software responds.

Nulls

Null means no value. Testing is carried out without providing any input. The software should be able to cope with this kind of situation where no input is provided to the software.

Example: Login page has two empty fields, Login ID and password. Imagine a test case where the user tries to log in by providing the login ID and leaves the password field blank or vice versa. Here, only one field is filled by the user and the other field is left blank or null.

Bad Data

This is similar to test-to-fail. The tester enters erratic or irrelevant data and checks the response of the software. The software might work correctly for all the correct inputs, but it is very important to check its response when incorrect or irrelevant data is provided as input.

Example: A login password field of a Webpage accepts only numerical values. It does not accept alphabets or combination of alphabets or numbers as password. If the tester enters these data, i.e., alphabets or combination of alphabets and numbers as password to test the Webpage, it is called as a bad data test.

3.2.4 State Testing

State testing refers to testing the software state. State is a mode or condition of the software at any given time when the software is running. State diagrams are used to indicate the state of the software. This indicates the actual working and the logical flow of the software.

State based testing is used for high level black box testing of programming languages like object oriented programming. Object oriented programming languages that have features like encapsulation

can be easily depicted using state diagrams. These state diagrams consist of states and transitions. Every program has two important states, the start state which indicates the beginning of the program and the end state which indicates the end of the program. In between these two states, a number of intermediate states represent the logical flow of the software. During the executions, the program flows from one state to another. This change of state is called state transition.

Example: The Windows paint program expresses itself in different states. When the application is opened, the cursor takes the pencil tool as the default state. When a different tool is selected, say an air-brush, the cursor changes its state to bring out the properties of the air-brush. The internal working of the software toggles every time the user selects a different tool. This change can be termed as state transition.


Notes

All software that is event driven makes use of state based techniques to analyze the program flow. For example, the GUI programs make use of state based techniques both at the design stage and at the testing stage.

The state diagram is a graph that depicts all the states and transitions taking place during the flow. The tester checks the states and transitions occurring at each level of the state diagram. The tester has to check each state and the transition that takes place, since every state is unique and each transition is a new operation that is performed on the current state to generate an output that is moving to the next state. This helps the tester to understand the flow of the software and detect the occurrence of the bugs based on states and transition.

Example: Let us consider a program where a transition has to happen from state A to B and from B to C. The tester checks if the transition happens from state A to C and from C to B. This incorrect transition will be considered as a bug in the software.

State based technique helps to obtain a model that depicts the behavior of a program very clearly. However, drawing a state diagram is a challenging task. Any software has many states and a transition associated with it. Depicting the same can be a complex process.

3.2.5 Random Testing and Mutation Testing

Many other techniques of software testing are being used by testers around the world to obtain efficient testing results. Two of the most popular and widely used testing techniques are random testing and mutation testing.

Random Testing

In random testing, the tester provides random inputs to the test software and checks the output of the software against the expected output. Any mismatch in the actual output and expected output will be treated as a bug.

Example: In a mathematical expression,
C = 2 * B;
The variable B can take values from -500 to 500. The test can be carried out for five random values- 24, -348, 75, 499, and -1. The expected result would be 48, -696, 150, 998, and -2 respectively. The expected result is compared with the actual result when this expression is executed.

Random testing is one of the popular testing techniques which is also called as Gorilla testing or Adhoc testing. This is discussed in detail in Chapter 9.

Mutation Testing

In this type of testing, the tester makes minor modifications in the program's source code and performs the test. If the program clears the test even after it is modified, the program is considered to be defective. Any such modification is called mutation. The main objective of mutation testing is to check

the behavior of the code when some modification is made to the original code. Depending on the behavior of the system, bugs can be found in the software, after the modifications are made to the original software. Mutation testing is carried after the test software has cleared the preliminary test and is found bug free. The mutation can vary from a simple change of operator name or variable name to replacing the logical operator with its inverse. A complex mutation involves changing the order of execution of the code or removing some lines of codes.

Example: The expression,
a !=b;
can be mutated as,
a ==b;

After the mutation, the software is executed to record the output of its behavior. The test cases used are those which have been created during preliminary testing. If the test cases are well written, the mutated program should fail. However, if it clears the test case, either the test case is weak which has to be re-checked or the program has bugs.

Case Study

Black Box Testing for Banking Applications

A global banking firm with over 30 million customers wanted to increase its business volume by concentrating more on customer retention and enhancing relationship with them. For this the bank used client-server applications for their banking software.

Every client server application needs continuous up-gradations to facilitate integrated transactions from common servers and reworking navigation and usability that meets business needs. Any failure or errors in such applications may lead to severe financial losses to the customers and the bank. This also affects the reputation of the bank. The bank provides various deposit and loan products, automobiles finance, housing loans and personal loans for their customers across the globe. Bugs in the software might cause problems related to availability, scalability, and performance. They have to be resolved before the application reaches the production environment.

A team of highly skilled testers were given the task of testing the banking application. Subject Matter Experts gave extensive knowledge and training to the team to make the testers aware of the bank's business requirements. The testing team developed a high level test plan, and designed business scenarios, which were extensive black box testing test scenarios to test the application. The team ran these test cases and scenarios in a configured environment with end-user perspective to detect critical bugs in the application.

Based on the test case results, the team suggested up-gradation in the application. Since the test cases were domain specific and end user oriented, the team was able to efficiently carry out the testing process. Therefore, with the help of extensive black box testing, the testing team was able to address all the concerns of the bank successfully.

This extensive black box testing helps to:

1. Measure the factors that affect the performance of the application.
2. Provide confidence to the bank in the application's performance before it is deployed in the actual working environment.
3. Establish performance standards for all the applications and monitor critical indicators to assess the change in the expected performance levels.

Questions

1. What was the problem that the bank faced? How was it addressed?
2. Do you think extensive black box testing helped the bank to resolve the issue that it was facing?

Adapted from (http:// www.lntinfotech.com/casestudies/testing/reduction_in_testing_costs.pdf)

3.3 Summary

- Structural testing techniques check the occurrence of bugs in the test software using the actual codes of the software. The tester works with the source code of the software while performing the test.
- In a functional test, the tester is not aware of the actual working of the software. The test analysis is performed based on the outputs that the software generates for various inputs. The bugs are detected by comparing the expected output with the obtained output.
- Black box testing is a functional testing technique. The tester performs the test to check the behavior of the software by providing pre-defined inputs and analyzing the outputs.
- A tester performing white box testing knows the actual code level working of the software. The test cases target to find the bugs associated with the code's logic, structure, module interface and memory organization.
- In order to make the software bug free, a certain level of both black and white box testing has to be performed on software.
- A static black box testing involves checking for bugs in the specification document. Any mistakes or incorrect information present in the specification is considered as a bug.
- Dynamic black box testing refers to testing for bugs by executing the software.
- Test to pass and test to fail are dynamic black box testing techniques. Test to pass involves providing normal inputs to the software to check whether it works without any bugs. During test to fail, the tester provides erratic inputs to check the software.
- Equivalence partitioning involves grouping similar test cases and performing the test where, a test case from each class is used to perform the test.
- Dynamic black box testing techniques like the data testing is carried out to check for occurrence of bugs in the input data provided to the software. State testing focuses on the transitions of internal software state.
- Other testing techniques such as random and mutation testing are some of the popular dynamic testing techniques to perform efficient software testing.

3.4 Keywords

Encapsulation: A technique where the internal representation of an object is generally hidden from view outside of the object's definition.

Neural Networking or Artificial Intelligence Simulation: This is a branch of computer science where intelligent machines are created through software programs that simulate or reproduce the creative functions of the human brain.

Source Code: It refers to a collection of statements or declarations that are written in a computer programming language. Source code needs the compiler or interpreter to translate the file into the object code before execution.

State Diagram: It is an illustration of the states an object can attain as well as the transitions between those states.

3.5 Self Assessment

1. State whether the following statements are true or false:
   (a) The testing strategies and techniques are developed to address a particular type of need or to test certain required parameters of software.
   (b) Static Black box testing consists of viewing the specification at the high and low level.
   (c) The testing team must strike a balance depending on the project requirement to adopt both black and white box testing to test the software.

(d) While performing software test the tester should first begin the test with test to fail and check whether the software works fine without any bugs.

(e) The tester enters erratic or irrelevant data and checks how the response of the software while performing mutation testing.

2. Fill up the blanks:

(a) During functional testing the tester checks only the _____ of the software and will not check the actual code.

(b) Static black box testing is performed to check the specification using _____ and _____ techniques.

(c) Developing efficient_____ is very essential during testing.

(d) _____ testing is used for high level black box testing.

(e) The main objective of _____ is to identify the test cases that perform same kind of testing and similar output.

3. Multiple Choice Questions

(a) Identify the testing technique that is used to test how the actual code works.

(i) Structural testing

(ii) Functional testing

(iii) Static testing

(iv) Black box testing

(b) Which of the following testing will help to expose any ambiguities or inconsistencies in the specifications and are carried out from a user's perspective?

(i) White box testing

(ii) Automation testing

(iii) Manual testing

(iv) Black box testing

(c) Which dynamic testing technique's main focus is to push the software to its limit and check the bugs that occur when the software is operated under extreme conditions?

(i) Test to pass

(ii) Test to fail

(iii) Data testing

(iv) State testing

(d) Which testing is also called as Adhoc testing?

(i) Equivalence Partitioning

(ii) Data testing

(iii) Random testing

(iv) Test to pass

Answers: Self Assessment

1. (a) True (b) True (c) True
   (d) False (e) False
2. (a) Behavior (b) High Level and Low Level (c) Test cases
   (d) State based (e) Partitioning
3. (a) Structural testing (b) Black box testing (c) Test to fail (d) Random testing

3.6 Review Questions

1. Do you agree with the fact that the logical flow of the software in its different forms (states) can be tested? If so, which type of testing will you apply?
2. "Static black box testing is more research oriented and the research helps to understand how the specification is organized and the reason behind the organization of the specification." Justify that high level and low level static black box testing improves quality.
3. What makes you think that Test to pass is different from Test to fail? Explain.
4. "Selecting the right testing method for testing the software is very important, since both black and white box testing methodologies have their merits". Could you list the merits of both black box and white box testing techniques?
5. Do you believe that there is a difference between boundary condition and sub-boundary condition? Explain.
6. Do you agree that Equivalence partitioning reduces the number of test cases without compromising the quality of the test being carried out? Explain
7. "Developing efficient test cases is very essential during black box testing." Why do you think so?

3.7 Further Readings

Books

Ron Patton, Software Testing-Second Edition, SAMS Publishing, USA
Hutcheson, Marnie L, Software Testing Fundamentals, Wiley Publishing, USA
Kassem A. Saleh, Software Engineering, J.Ross Publishing, 2009, US

Online link

http://qastation.wordpress.com/2008/04/21/static-testing-vs-dynamic-testing/
http://www.adager.com/vesoft/automatedtesting.html
http://www.scribd.com/doc/2453259/Testing-Techniques-and-Strategies
http://www.testinggeek.com/index.php/testing-articles/137-equivalence-partitioning-introduction
http://www.cc.gatech.edu/classes/cs3302_98_summer/7-02-unittest/sld009.htm
http://www.slideshare.net/nworah/types-of-software-testing
http://www.ece.cmu.edu/~koopman/des_s99/sw_testing/
http://www.softwaretestinghelp.com/what-is-boundary-value-analysis-and-equivalence-partitioning/

Unit 4: White Box Testing

CONTENTS

Objectives

After studying this unit, you will be able to:

- Explain static white box testing
- Explain dynamic white box testing

Introduction

The IEEE definition of software lists four components which are needed in order to assure the quality of software applications: computer programs or the code which is the brain behind any application; procedures that define the flow of the program, its methods and the way of functioning; documentation needed for developers and users; the data that includes parameters. The computer programs or the source code of the software is an important artifact that needs to be tested for ensuring quality of the software product. The testing that encompasses the verification of the computer programs and the logic of the application is known as White Box testing.

IEEE defines White box testing as "The testing that takes into account the internal mechanism of a system or component". White box testing takes care of the intricacies of the product and evaluates it for accuracy and precision, to meet the requirement specifications.

White box testing verifies the designs and codes involved in the development of a software product. It involves validating whether the code has been implemented as per design specifications and also validating the security concerns of the software's functionality. Thus skilled testers with knowledge of programming are required to conduct white box testing.

White box testing helps the software tester to find out the correct type of input data used to test the application effectively, i.e. the tester is aware of the internal coding and hence is able to optimize the code. When the tester removes the extra line of code, it enables the testers to find the hidden defects. According to Pressman, white box testing helps a software tester to perform the following functions:

1. Test independent paths within a unit or a module.
2. Test the logical correctness (test both the true and false conditions).
3. Test loops, specifically at their boundaries and check the operational boundary correctness.
4. Test internal data structures to ensure their validity.

White box testing provides greater stability and reusability of test cases. The software application is tested in a thorough way and thus raises the customer satisfaction and confidence.

Did you know? White box testing is also referred to as glass box testing or structural testing or open box testing or clear box testing, due to its nature of examining the internal workings.

4.1 Static White Box Testing

Static white box testing methodology involves testing the internal logic and structure of the code without compiling and running the program. The main advantage of performing static white box testing is that bugs are found early and those that cannot be uncovered by dynamic white box are also identified. To perform this type of testing, a software tester needs to have the knowledge of software coding and the internal logic. This enables a software tester to differentiate between the statement or the path which works from those which do not work.

Static white box testing involves a procedure of analyzing data flow, control flow, information flow, and also testing the intended and unintended software behaviors. There are different aspects that are examined which include codes, branches, paths, and internal logic. Let us now discuss the various methods by which static white box testing is performed.

4.1.1 Examining the Design and Code

Examining the design and code refers to examining and reviewing the codes without execution. We have three methods through which bugs are identified and captured. They are reviews, inspections, and walkthroughs.

The three methods use procedures and error detecting techniques for analyzing the bugs found and also subsequently correct them. There are planned review meetings that are held and the developers and the testers discuss the nature of the application and the probable areas of defects. When the areas are identified, they are either corrected or marked for dynamic white box testing.

Did you know? Static white box testing is also referred to as structural analysis.

Notes

The responsibility of performing static white box testing varies from one development team to another development team. In some organizations, the programmers organize and run reviews by inviting the testers as observers, while in some organizations the testers perform the task by asking the programmers, who wrote the code along with other peers, to help them in their reviews.

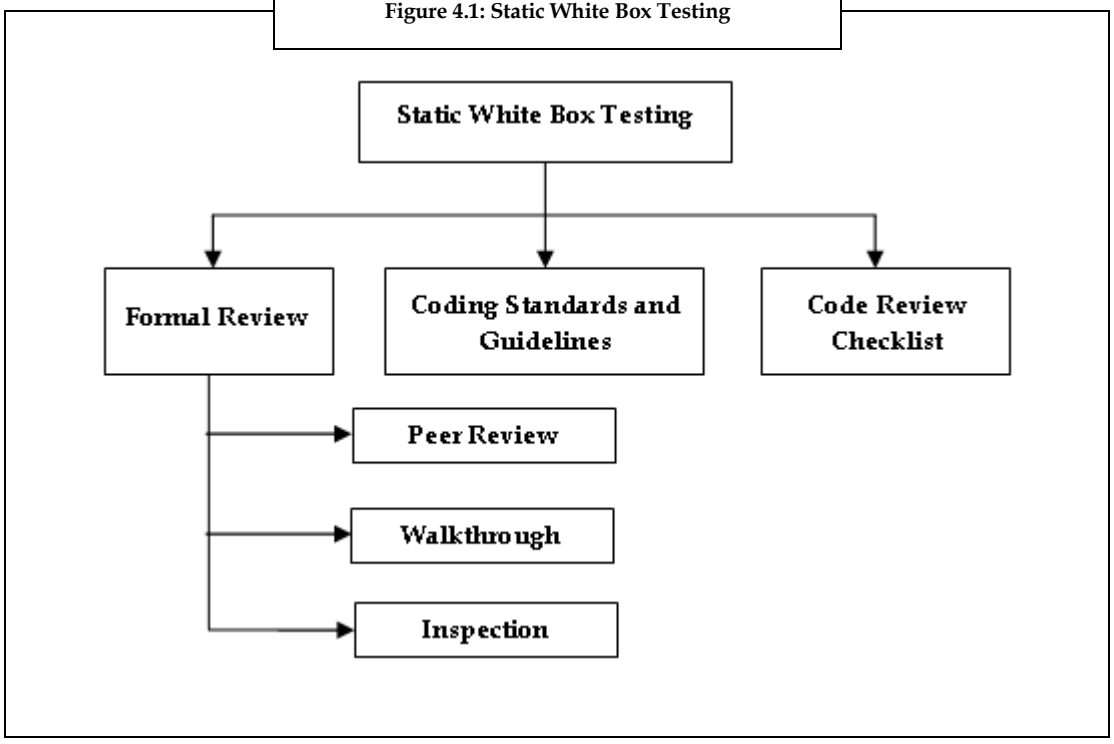
Static white box testing is found to be a cost-effective method of testing. The advantage of performing a static white box testing is that it provides the testers with better ideas about the test cases while implementing them during software development.

Programmers write codes and a few peers conduct reviews on them. The development team conducts the static white box testing and reviews the results. In addition, the development team also invites testers to observe the process of testing.

Static white box testing is seldom carried out in the software testing process, as there is a misconception that it is time consuming, expensive and not productive when compared to the other alternative testing methodologies. Although, it is a formidable task, today organizations have realized the importance of testing and have started to hire and coach both testers and programmers in the field of white box testing.

The figure 4.1 depicts the diagrammatic representation of static white box testing.

Figure 4.1: Static White Box Testing



4.1.2 Formal Review

Formal reviews are carried out in static white box testing which involves formal meetings between the programmers and testers (or between programmers). In this meeting, there will be discussions pertaining to inspection of the software's design and code. The formal reviews are considered to be the first nets that capture bugs, since, prospective defect areas are discussed in these meetings.

For a successful formal review, four essential elements are required, and they are:

1. ***Identify Problems:*** A formal review must identify the problems related to design and code of the software. Participants are required to avoid criticism and carry a positive attitude and must be diplomatic and not emotional about reviews.
2. ***Follow Rules:*** Formal reviews follow a fixed set of rules. The number of lines of code to be reviewed per day along with the time spent on the job is to be strictly adhered to. This is done so that reviews can be carried out more smoothly.
3. ***Prepare:*** Formal reviews expect each participant to prepare and contribute towards the meeting. Based on the type of review, each participant's role differs. The problems discovered in the review process are generally found during the preparation process and not actually during the review.

4. ***Write a Report:*** The formal review group produces a written report summarizing all the results of the review and the report is made available to the rest of the development team. Thereby, the problems encountered are shared with the team.

There are many steps involved in formal review. They are peer reviews, walkthroughs, and inspections.

Peer Reviews

Peer reviews are the informal reviews where team members conduct reviews amongst themselves. They are also known as buddy reviews.

Peer reviews are conducted with a programmer who has been involved in designing the architecture or code along with other programmers or testers, who act as reviewers. To ensure an effective review, the participants involved in the review are required to adhere to the four key elements of formal review (identify problems, follow rules, prepare and write a report).

Walkthroughs

Walkthroughs are the second step of the formal reviews. In this method, the programmer who developed the code presents the code to a group consisting of five or six member team of programmers and testers. A walkthrough is conducted to provide an overview about the structure of the code in the presence of a senior programmer and other reviewers.

The presenter reads through the code line by line, or function by function and explains what each function and line of code means. Relevant comments and queries are addressed during the walkthrough session. Since the number of participants in a walkthrough is more than those in the peer review session, it becomes even more important to follow rules and have periodic follow-up meetings.

After the completion of the review, the presenter makes a report of the meeting and also the way the bugs were addressed.

Inspections

Inspections follow a structured format. It is very different from walkthrough and peer reviews. The person who presents a code is not the real programmer. The participants are called inspectors. The inspectors are provided the task of reviewing the code from the user's and tester's perspective. This helps in bringing out the views about the product from various perspectives, thus helping in identifying the different bugs in the product.

The inspectors are provided with the task of reviewing the code backwards i.e. from end to beginning. This is done to ensure that the product has been evenly reviewed. Inspectors are also assigned the task of moderators and recorders to ensure that the testing is adhering to the rules and effectively running the reviews.

After the completion of the inspection, another meeting is conducted by the inspectors alone to discuss the defects that were found and they work with the moderators who are competent programmers to identify the areas of rework. The programmer then rectifies the defects and the moderators verify the same to ensure that it is done properly. Re-inspections are conducted based on the criticality of a software bug that is found.

4.1.3 Coding Standards and Guidelines

In the formal review method, inspectors look only for the problems and omissions in the code. Bugs are however found by carefully analyzing the code which is done by the senior programmers and testers.

Sometimes, there are also possibilities where a code may operate correctly but may not be written to meet the specification standards. This is similar to writing English which is grammatically correct, but may not convey the correct meaning.

To handle such situations, some standards are fixed based on the have-to-follow rules of Do's and Don'ts. Along with them, some guidelines are also prepared. Guidelines are the best practices and recommendations which are preferred to be followed. Standards are rules which must be adhered to, whereas guidelines are instructions which enable a person to follow a set of standards.

The three reasons for adhering to standards and guidelines are:

1. ***Reliability:*** It has been observed that a code which is being written for a particular standard with formal guidelines is more reliable and secure than the ones that are not.
2. ***Readability or Maintainability:*** Codes which have been written based on standards and guidelines are easier to understand and maintain, when compared to the ones which are not.
3. ***Portability:*** Codes written by programmers must be portable enough to run on different hardware and also different compilers. When standards and guidelines are followed, it becomes easier for people to access the code. Sometimes, project requirements may demand to meet the international standards and guidelines.

Hence, it is necessary to have a standard and set of guidelines for programming and ensuring verification in a formal review. Improper usage of statements can result with lot of bugs in a system.

4.1.4 Code Review Checklist

Code reviews are performed in addition to the general process of comparing the code against the standards and guidelines. This ensures that the design requirements of the software project are met. To conduct code reviews in detail, some amount of programming experience is required. The following example shows some of the code review questions.

Example: Does the code do what it has been specified in the design specifications?

Does the software module have another similar existing module, so that it could be reused?

Does the module have a single entry point and single exit point (As multiple entry and exit points can be tedious to test)

We will now discuss the various errors that are discovered while testing. They are:

1. ***Data Reference Errors:*** Data reference errors relate to the errors which are caused due the usage of variables, constants, arrays, strings, or records which are not properly declared or initialized to use and refer them.

   Some of the points which you need to remember while looking for data declaration errors are:

   (a) Check if any un-initialized variables are referenced

   (b) Check if the arrays and the string subscripts integer values are within the array's bounds or string dimension

   (c) Check if there are any "off-by-one" errors in indexing operations or references to arrays

   (d) Check if a variable is used where a constant would work better

   (e) Check if a variable is assigned a value that's of a different type than the variable

   (f) Check if memory is allocated for referenced pointers

   (g) Check if the data structures are referenced in different functions defined identically

Caution Data reference errors are the primary cause for buffer overruns - the main bug concerned with security issues.

Task Consider a scenario, where you have been assigned the task of checking the security of logging into a Gmail account. Prepare a set of security code review questions for this scenario.

2. ***Data Declaration Errors:*** Data declaration errors occur due to improper declaration of variables and constants.

   Some of the common points to ponder on while checking for data declaration errors are:

   (a) Check if the variables are assigned the correct length, type, storage class

Example: A variable that is incorrectly declared as an array instead of a string.

   (b) Check if a variable is initialized at the time of declaration, and also analyze if it is properly initialized and consistent with its type

   (c) Check if there are any variables with similar names

   (d) Check if there are any variables declared which are never referenced or just referenced once (should be a constant)

   (e) Check if all variables are explicitly declared within a specific module

3. ***Computation Errors:*** Computational errors arise due to errors in calculations -- where the expected results are not obtained due to erroneous calculations.

   Some of the questions pertaining to computational errors are:

   (a) Check if any calculations use variables which have different data types

Example: Adding integers and floating point numbers

   (b) Check if any calculations use variables which have the same data type but vary in size

Example: Adding long integers to short integers

   (c) Check if the compiler's conversion rules for variables of inconsistent type or size understood and considered while calculating

   (d) Check if overflow or underflow in the middle of a numeric calculation possible

   (e) Check if it is ever possible for a divisor/modulus to be zero

   (f) Check if a variable's value goes outside its meaningful range

Example: The probability of a result being less than zero percent or greater than 100 percent.

   (g) Check if the target variable of an assignment is smaller than the right-hand expression

   (h) Check if parentheses are needed for clarification

Example: For expressions containing multiple operators, there is confusion about the order of evaluation.

   (i) Check if for cases of integer arithmetic, the code handles some calculations, particularly division, which results in loss of precision

4. ***Comparison Errors:*** Comparison errors occur during boundary conditions such as 'less than, greater than, equal, not equal, and true or false'.

   Some of the common points to be analyzed for comparison errors are:

   (a) Check if the comparisons are correct

Example: Using < instead of <=

(b) Check whether there are any comparisons between floating-point values

Example: Checking for precision problems when comparing.

(It is similar to checking whether 7.00000007 is close enough to 7.00000008 to be considered equal)

(c) Check whether the operands of a Boolean operator are correct

Example: Checking, if an integer variable containing integer values is used in a Boolean calculation

In the 'C' language, zero is considered as true and non-zero is considered as false.

(d) Check if each Boolean expression states what it is supposed to state, works as expected and if there are any doubts about the order of evaluation

5. ***Control Flow Errors:*** Control flow errors arise due to improper behavior of loops and control structures in a language. They are caused by direct or indirect computational or comparison errors.

Some of the points to be analyzed while checking for control flow errors are:

(a) Check whether each switch statement has a default clause

(b) Check if the nested switch statements are in loops

(c) Check the possibility of a loop not executing

(d) Check if the compiler supports short-circuiting expression in evaluation

(e) Check if the language contains statement groups such as begin...end and do...while, are the 'end's explicit and do they match their appropriate groups

(f) Check if there is a possibility of premature loop exit

(g) Check if there are any "off by one" errors which may cause unexpected flow through the loop

6. ***Subroutine Parameter Errors:*** Subroutine parameter errors occur when incorrect data is passed to and from subroutines.

The following are the points to be noted while checking for subroutine parameter errors:

(a) Check if constants are passed to the subroutine as arguments or are they accidentally changed in the subroutine

(b) Check whether the units of each parameter matches with the units of each corresponding argument

(c) Check the types and sizes of the parameters received by a subroutine match with those sent by the calling code

(d) Check if a subroutine alters a parameter that's intended only as an input value

7. ***Input/Output Errors:*** Input/output errors relate to errors such as reading from a file, accepting input from a keyboard or mouse, and writing to an output device such as a file or screen.

The points to be analyzed while checking for input/output errors are:

(a) Check whether the software strictly adheres to the specified format of the data being read or written by the external device

(b) Check if the error condition has been handled

(c) Check if the software handles the situation of the external device being disconnected

(d) Check if all the exceptions are handled by some part of the code

(e) Check if all error messages have been checked for correctness, appropriateness, grammar, and spelling

8. ***Other Checks:*** The ones which do not fit into the above mentioned categories fall under the other checks. In this category of error checks, we will check for the following:

(a) Check if the code is portable to the other OS platforms

Example: The GCC compiler warnings(GCC is the GNU Compiler Collection used to compile C programs)

(b) Check whether the code handles ASCII and Unicode

Example: Checking whether the software will work well with languages other than English.

(c) Check whether your code passes the lint test

(d) Check issues related to internationalization

(e) Check whether the code relies on deprecated APIs

(f) Check whether the code ports to architectures with different byte orderings

4.2 Dynamic White Box Testing

Dynamic white box testing is a validation check which involves testing and running the test cases with input values. The dynamic white box testing ensures that the application works according to the specification throughout the execution. The results of the execution give proof about the actual behavior of the code. A software tester collects ample information which enables the tester to analyze what the code does, how it works, what to test, what not to test, and thus determines the approach of the testing process.

Notes

Dynamic white box testing is also known as structural testing, as it enables the tester to view the structure of the code and its design and run the tests.

The four prominent areas that dynamic white box testing encompasses are:

(a) Testing low-level functions, procedures, subroutines, or libraries directly.

Example: The low-level functions, procedures, and subroutines are known as API's (Application Programming Interfaces) in Microsoft.

(b) Testing the software at the top level, as a completed program, by adjusting the test cases based on what we know about the software operation.

(c) Gaining access to read variables and state information from the software and enabling the tester to determine whether the tests are doing what they were designed to do. Forcing the software to do things that would be difficult, if they are tested normally.

(d) Measuring how much of the code and which particular code a tester "hits" when he/she runs the tests and also how he/she adjusts the tests to remove the redundant test cases and add the missing ones.

4.2.1 Dynamic White Box Testing vs. Debugging

Dynamic white box testing and debugging are two techniques which appear similar, since they deal with the code and the bugs in software. They do overlap when bugs are found. However, they are different in their goals. Dynamic white box testing finds the bugs in the software while debugging deals with the process of fixing the bugs that have been found during the testing phase.

As a software tester, you are required to emphasize on the test cases that demonstrate a bug. In white box testing, you narrow down to the information pertaining to the number of lines of code which look erroneous. The programmers debug the erroneous lines of code which are reported to them. They find the reason for the cause of bug and fix it.


Notes

Most often, while performing low level testing, a software tester uses the same tools which a programmer uses.

A code-level debugger is used to watch variables and set break conditions.


Example: When a program is compiled, the same compiler is used with different settings to ensure that major defects or errors are detected.

4.2.2 Testing the Pieces

In dynamic black box testing, a bug is found at the final stages of testing. It is difficult and at times impossible to figure out the occurrence of bug and hence a tester will be unable to learn which part of the software is causing the problem.

There are also possibilities of some bugs which hide the existence of other bugs and these bugs might cause the software to fail. Although, a programmer fixes the problem, when the tester re-runs the software, it will fail again. As a result, bugs keep accumulating and it becomes impossible to get to the core of the defect.

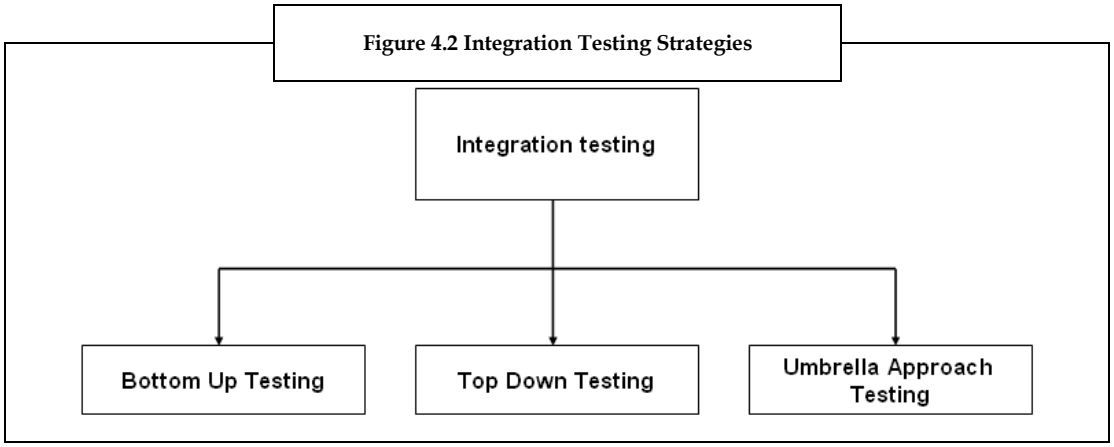
Unit and Integration Testing

To overcome the problem of accumulating core defects, the concept of testing the software in pieces and gradually integrating these pieces into larger portions to form the system came in to existence. Thus, the lowest level of testing known as unit or module level testing came up.

In this method, the small units are tested where bugs are logged and fixed. These units are then integrated and integration testing is performed against groups of modules. This process continues until a complete system is ready for testing. This is known as system testing. The unit and integration testing enable the tester to locate the bugs in software at a lower level, thereby reducing the cost. The bugs found in the multiple units while integrating relate to how modules interact with each other.

The three integration testing strategies are shown in figure 4.2.

Figure 4.2 Integration Testing Strategies



Now, let us discuss the three strategies of integration testing.

1. ***Bottom Up Testing:*** The bottom up testing is conducted from the sub module to the main module. In the absence of the main module (when the module is not developed), a temporary program known as 'Drivers' (Drivers are the written code that are used to execute the modules that are being tested. These drivers act as real modules by sending test-case data to the modules, reading the results and verifying their correctness) to simulate the main module. This approach helps in creating easy test conditions and easy observation of test results.
2. ***Top Down Testing:*** The top down testing is conducted from the main module to the sub module. In the absence of the sub module (when the module is not developed), a temporary module known as 'Stub' (Stubs are small pieces of code which act as interface modules by feeding "fake" data) is used to simulate the sub module. This type of testing boosts the morale of the program by eliminating the flaws. The top down method of testing relates to testing all the higher-level modules by using stubs as low-level interface modules. This type of testing is found to be very useful, as we can quickly run through the numerous test values and also validate the operations of a module.
3. ***Umbrella Approach Testing:*** The umbrella approach testing is a combination of both, the bottom up and top down testing approaches. This method focuses on testing the modules which have high degree of user interaction. The input modules are integrated in the bottom up approach and the output modules are integrated in the top down approach. It is considered to be a beneficial approach for the early release of GUI (Graphical User Interface) based applications used for enhancing the functionality.

   The umbrella approach of testing ensures:

   (a) Early release of limited functionality.

   (b) Reduction in the need for stubs and drivers.


Notes

The only disadvantage of umbrella approach is that it leads to more regression testing as it follows a systematic approach of testing.

4.2.3 Data Coverage

Data coverage focuses on data which is tracked completely through the software. It can relate to the values stored in variables, constants, arrays, and data structures. It includes the keyboard and mouse input, files, screen input and output and those that are sent to modems, networks, and so on.

In the data coverage method, the code is divided into data and states, relating to program flow which is similar to black box testing. Hence, it becomes easier to map the white and the black box test cases.

Data coverage deals with the aspect of generating test data sets which cover all the required tests before the software component is certified as reliable and shipped to the customer.


Notes

The statement coverage testing was found to be the cheapest method of testing as it produced the maximal effect for the smallest number of tests applied, but this was an ineffective technique as the number of errors uncovered were very less.

The types of data coverage testing are:

1. ***Data Flow:*** Data flow involves tracking the data through the software. During the unit testing phase, tracking is done only in the small modules, as tracking all the modules in a system becomes a tedious process. When a function is tested at the low level, a debugger along with a watch variable is used to view the program during execution. However, in the white box testing, one cannot view the program execution, but can only view the value of the variable at the beginning and at the end. The dynamic white box testing gives the provision of checking the

intermediate values during program execution. This enables a tester to analyze things better, by viewing the changes in a program.

2. ***Sub-Boundaries:*** Sub-boundaries in software are the most common places where bugs can be found.

Example: An operating system which is running on low RAM may move the data to a temporary storage on the hard drive. In this case, the boundary may not be fixed, as it depends on the space remaining on the disk. There are more possibilities of errors occurring at the sub-boundaries.

Notes

While performing white box testing, code must be carefully examined around the sub-boundary conditions and test cases should be created accordingly.

3. ***Formulas and Equations:*** Generally formulas and equations are not visible and are embedded deep in the code. Hence, we cannot know their effect and presence.

4. ***Error Forcing:*** This is a method of data testing, where a software tester can force a variable to hold a value in order to check how the software handles it. When the software runs in a debugger, there is a privilege of not just watching the variable and the values that they hold, but you also have the privilege of adding required specific values. (A debugger is used to force a variable value to zero)

Caution

While using error forcing, one must ensure that only situations which happen in real world scenarios are created. If not, an invalid test case would be created.

4.2.4 Code Coverage

Testing a black box is just the half work done and the rest needs to be completed with the white box testing. While testing, testers need to assure that they test the entry and exit of every module, every line of code, and follow logical and decision path of the product. This methodology of testing followed is known as code coverage testing.

Code coverage testing is a dynamic white box testing where testing is done by executing the test and the tester has complete access to the code. He/she is also aware of the parts of the software that have passed and the parts that have failed.

Code based testing, which involves detecting errors by following execution oriented methods, is also known as white box testing. Here, a tester is expected to know and understand the low level design and identify the code-based approach and to apply the techniques in the code that is written.

The strategy of white box testing is to ensure that the internal structure of the program has a logical flow. The need for code-based testing is to:

(a) Understand the objective and arrive at test cases.

(b) Check programming styles and compliance with coding standards.

(c) Check for logical errors and incorrect assumptions introduced during coding.

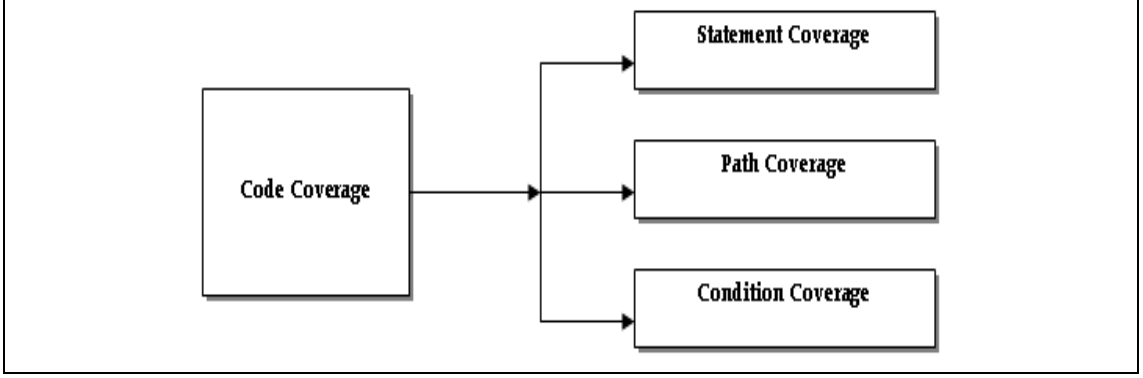
(d) Find incorrect assumptions about execution of paths.

(e) Find typographical errors.

A tester is also required to maintain a checklist for code walkthrough, code inspection, traceability matrix, and record bug findings in the bug report and in the review records.

The different types of code coverage testing is shown in figure 4.3

Figure 4.3: Code Coverage Types



Now, let us discuss the types of code coverage.

1. ***Statement Coverage:*** The simplest form of code coverage is known as statement coverage or line coverage. A software tester needs to ensure that all the statements are executed at least once.

Example:

1: PRINT "Good Morning"

2: PRINT "Be Positive"

3: PRINT "Today is a Great Day"

4: END

Did you know? Data coverage method is considered to be more effective than the statement coverage testing, as it uncovers more faults with fewer tests.

In the above mentioned program, all the statements are executed once. Sometimes statement coverage can be misleading. This is due to the fact that statement coverage can provide information relating to the execution of each statement only, and not relating to the paths traversed in the software.

2. ***Path Coverage:*** Path coverage relates to covering all the paths in the software. The simplest form of path testing is known as branch coverage testing.

   Loops are the important parts of a structured program. The direction in which the control proceeds in a program and the number of times the statement gets executed is based on the initiation and termination condition of loops. Testing such types of loops is a challenging task, as an error may be revealed only after exercising some of the sequence of decisions or particular paths in a program.

Example: An 'If statement' that creates a condition

1: PRINT "Good Morning"

2: IF Date$ = "25-12-2008" then

3: PRINT "MERRY CHRISTMAS"

4: END IF

5: PRINT "The date id:"; Date$

6: PRINT "The time:";Time$

7: END

The above program can be run on a single test case with the date set to December 25, 2008. This program will execute all the statements. Although we have tested each statement, the branches were not tested. Thus, code coverage analyzers will account for code branches and report both the branch coverage and statement coverage. It also provides details about test effectiveness.

3. ***Condition Coverage:*** Condition coverage is a complicated path testing. This is done by adding an extra condition to the IF statement. Condition coverage is a combination of branch coverage and more detailed conditions. Branches alter the flow of sequence in a program and jump from one part of the program to another. Both conditional and unconditional transfers can occur. In this type of coverage, every branch of the control flow graph (A control flow graph is a graphical representation of all the paths which the program has traversed through during execution) is executed at least once and the values of the conditions are also exercised at least once. The simplest condition criterion called the basic or elementary condition coverage requires each elementary condition to be covered, which implies that each elementary condition should have both True and False outcomes at least once during the execution of test set.

$$\textit{Basic Condition Coverage} = \frac{\textit{Number of executed conditions}}{\textit{Total Number of conditions}}$$

Example: Multiple conditions in the 'if statement 'can create more paths through the code.

1: PRINT "Good Morning"

2: IF Date$ = "25-12-2008" AND Time$ = "00:00:00" THEN

3: PRINT "MERRY CHRISTMAS"

4: END IF

5: PRINT "The date is: "; Date$

6: PRINT "The time is: "; Time$

7: END

In line 2 in the above program, both date and time are checked. Thus, condition coverage takes the extra conditions. We will require four test sets of test cases which assure that the each IF statement is covered.

Test cases for a Full Condition Coverage

Date$ Time$ Line # Execution

25-12-200711:11:11 1, 2, 5, 6, 7

25-12-2007 00:00:00 1, 2,5,6,7

25-12-2008 11:11:11 1, 2,5,6,7

25-12-200800:00:00 1,2,3,4,5,6,7

All the four cases are considered to be important, since they traverse through different conditions of the IF statement such as False-False, False-True, True-False, and True-True.


Task

Write the statement coverage test cases for the following program.

```
Void function eval (int X, int Y, int A)
{
If (X>1) and (Y=0)
then A=A/X;
if(X=2) or (A>1)
then A=A+1;
}
```

(Note: X=2,Y=0, A=3 (A can be any assigned value)


Task

1. Based on statement coverage, write a simple program to print the day, date, time, month and year.
2. Based on condition coverage, write a simple program to print the marks you obtained in each subject and the class to which those marks belong. (If mark is greater than 60 implies class 1, if marks greater than 50 but less than 60 implies class 2, if marks less than 50 but above 35 implies class 3, if marks less than 35- implies fail)


Case Study

Secure Online Transactions

Myway is a newly established company which wanted to start their online sales and hence was involved in the development of an online e-commerce Web site. They wanted a Web site which would facilitate online transfer of funds. Myway had outsourced its payment processing to a third-party Internet enabled financial transaction payment firm.

The third party payment software had several customized interfaces to enable an easy payment process between the customers and Myway's account.

A high-level security risk analysis was conducted on the system to check the security measures of the third party software. The assessment identified several transactions between the payment interfaces and the application. A fraudulent transaction would have serious impact on both the customers and the company ('Myway'). Some customers might also face financial hardships due to the unauthorized transactions which may deplete an account. Such situations can damage the reputation of the company.

In view of such circumstances, a white box testing was carried out on all the modules which were using the payment interfaces. The following steps were followed:

The component interfaces were identified.

1. Trust relationship boundaries were recognized along with the component interactions.
2. The data flows between the components were monitored.

Abuse test cases (These cases describe the ways in which the application can be misused) were developed and tested. One of the abuse test cases was to create a payments processing functionality as an anonymous user. The trust relationship mapping and data flow showed a path where inputs from the users were not validated or authenticated.

 Contd…

A test case was developed to check for an account transfer from an external account to the company's account anonymously. In addition, the account and the transaction went through successfully. This indicated a critical software failure.

Thus, risk assessment identified a weak authentication component in the payment component of the system. Trust relationship boundaries and data-flow analysis were also conducted on the component. After thorough testing, it was proved that a hacker could easily gain access to the company's administrative accounts where the hacker could redirect transactions from Myway's account to another.

We can conclude that by performing risk analysis and white box testing in specific areas can quickly aid in revealing the design assumptions and implementation errors. Hence, it is necessary to collect information about the various bugs in software and analyze them.

Conclusion:

White box testing for security is very necessary and must follow a risk-based approach to balance the testing effort. Architectural and design-level risk analysis provides the right plan to perform white box testing.

White box testing can be used with black box testing to improve overall effectiveness of the software tested, by uncovering programming and implementation errors.

Questions

1. Identify the steps carried out in white box testing for the payment modules.
2. Emphasize the importance of white box testing and discuss abuse test cases.

Adapted from (http://basicqafundamentals.blogspot.com/2011/01/case-study-for-white-box-testing.html)

4.3 Summary

- White box testing gives a better understanding of the intricacies in the software product.
- The aim of the white box testing is to ensure that the internal mechanisms of the product work properly. It helps in code optimization.
- White box testing helps a software tester to analyze the design and code of the software and helps to have a better understanding of both the black box testing and the white box testing.
- Formal reviews are the formal meeting conducted amongst the programmers to discuss the design and the code of the software.
- Formal reviews are considered to be the first net, where bugs are captured.
- Peer reviews are conducted amongst the programmers and testers to review each other's work
- Walkthrough is the second stage of formal review, where the programmer explains the meaning of the code; line by line or function by function to a team of four to five members.
- Inspections are the most formal process of reviewing, which follow a structured format.
- The participants in the inspection process are called inspectors and perform a task of reviewing the code backwards, from end to beginning, to ensure that the product has been evenly reviewed.
- The three reasons for adhering to standards and guidelines are Reliability, Readability or Maintainability and Portability.
- Static white box testing, which examines the code, is considered to be the best means to find errors early.
- Static analyzers are available in the market to automate the testing approach.
- Compilers are improved by enabling their level of error checking and this helps them to find the errors in the code review.

- Dynamic white box testing approach greatly reduces the process of testing by providing the inside information about, what needs to be tested in a software.
- This approach helps the software tester to remove redundant test cases and adds the required test cases appropriately, thus improving the effectiveness of testing.
- Practice and experience as to what to test, when to test and how to test helps a software tester to become a good tester.

4.4 Keywords

ASCII: American Standard Code for Information Interchange (ASCII). It is the representation of alphanumeric numbers ranging from 0-127.

Deprecated: Disapproved code which might not exist in the next version.

Driver: It is a software module used for invoking other modules which are under test. They provide test inputs, monitor execution, and report the test results.

GCC: It is the GNU Compiler Collection and refers to a suite of tools. It is commonly used in compiling C programs. GNU is the name chosen because GNU's design is Unix-like.

Lint Test: Test lint is a coding advisor for unit testing and the process is known as lint test.

Off-by-one: An avoidable error where a loop iterates many times or very few times than what was designed.

Stub: It is a software program which substitutes for a software module which is defined elsewhere in a software system.

Unicode: Unicode is a computing standard and is used for consistent encoding.

4.5 Self Assessment

1. State whether the following statements are true or false:
   (a) White box testing is also known as clear box testing or glass box testing.
   (b) Static white box testing does not require skilled testers to perform testing.
   (c) White box testing is an inexpensive method of testing.
   (d) A formal review is considered to be the first net that captures the bugs.
   (e) The problems discovered in the review process are generally found during the preparation process and not during review.
   (f) The participants in a walkthrough are more than the participants in the peer review.
   (g) Re-inspections are conducted based on the criticality of a software bug.
   (h) Standards are structured and must be adhered to.
2. Fill in the blanks:
   (a) A software tester is required to think from the_____ point of view while performing testing.
   (b) White box testing is performed to ensure that the system adapts to malicious attacks and _____.
   (c) The three phases of static white box testing are _____, _____ and _____.
   (d) Walkthrough is the _____ step of formal review.
   (e) The inspectors are provided the task of reviewing the code from the _____and _____ perspective
   (f) Guidelines are _____ to be followed which are not really mandatory.

3. Multiple Choice Questions:

(a) The alternative name for white box testing is

(i) Structural analysis

(ii) Gray box testing

(iii) Glass box testing

(iv) Functional testing

(b) Which of the following stages do inspections come under?

(i) Peer review

(ii) Formal review

(iii) Code coverage

(iv) Data coverage

(c) From the following, select the choice which is appropriate to standards and guidelines

(i) Reliability

(ii) Scalability

(iii) Usability

(iv) Compatibility

(d) Which of the following refer to data declaration errors?

(i) Improper declaration of variables and constants

(ii) Improper declaration and initialization while referring and using variables

(iii) Improper calculations

(iv) Improper behavior of loops and control structure

(e) In which of the following testing types, the size of the collection is tested?

(i) Data coverage testing

(ii) Code coverage testing

(iii) Error forcing testing

(iv) Inspections

(f) Which of the following is a strategy of integration testing?

(i) Bottom-up testing

(ii) Bing Bang testing

(iii) Static white box testing

(iv) Black box testing

4.6 Review Questions

1. Are Debugging and Dynamic white box testing one and the same? Discuss their similarities and differences.
2. "Static white box testing is cost-effective." Justify
3. "There is a big difference between inspections and the other types of reviews." Substantiate.
4. Do you think that bugs could be caught through Formal reviews? How could we bring in code efficiency through the different methods of Formal reviews?
5. "The umbrella approach testing is a combination of both bottom up and top down testing approaches." Do you believe Umbrella approach of integration testing is useful?
6. "In the data coverage method, the code is divided into data and states " Explain
7. Justify the use of stubs and drivers in testing.
8. "Code coverage testing is a dynamic white box testing" Justify the same.
9. "White box testing has its disadvantages." Do you agree?
10. "Condition coverage is a complicated path of testing." Justify with an example.
11. Discuss the occurrence of errors in testing and the need for code review checklist.
12. Should coding standards and guidelines coexist during testing software? Justify.
13. Do you believe 'Testing the pieces' in software is advantageous?
14. Mention the importance of the four prominent areas of dynamic white box testing.
15. Given below is a java statement code in the correct and incorrect code. Write test cases that bring out the mistakes of the code:

```
/Correct Code

class TestProg {
   public static void main(String[ ] args)
  {

    int mark = 76;
    char grade;

    if (mark >= 90) {
   grade = 'A';
   } else if (mark >= 80) {
     grade = 'B';
   } else if (mark >= 70) {
     grade = 'C';
   } else if (mark >= 60) {
     grade = 'D';
   } else {
     grade = 'F';
   }
    System.out.println("Your Grade is:" + grade);
  }
 }

//Incorrect Code
```

```
class TestProg {
public static void main(String[ ] args)
{

int mark = 76;
char grade;

 if (mark >= 90) {
 grade = 'A'
 } else if (mark >= 80) {
   grade = 'B';
 } else if (mark >= 70) {
   grade = 'C'
 } else if (mark >= 60)
   grade = 'D';
 } else {
   grade = 'F';
 }
 System.println("Your Grade is:" + grade);
}
}
```

Answers: Self Assessment

1. (a) True (b) False

   (c) True (d) True

   (e) True (f) True

   (g) True (h) True

2. (a) Hacker's (b) Regular software failures

   (c) Formal Reviews, Coding standards and guidelines, Code review

   (d) Second (e)Users, testers (f) Instructions

3. (a) Open box testing

   (b) Formal review

   (c) Reliability

   (d) Improper declaration of variables and constants

   (e) Data coverage testing

   (f) Bottom-Up testing

4.7 Further Readings


Books

Ron P (2006), Software Testing-Second Edition,USA, SAMS Publishing

Marnie H L (2003),Software Testing Fundamentals, USA.Wiley Publishing

Kassem. A. (2009), Software Engineering.USA, J. Ross Publishing

Srinivasan D & Gopalaswamy R (2006), Software Testing: Principles and Practice USA, Dorling Kindersley.


Online link

http://www.testinggeek.com/index.php/testing-types/system-knowledge/50-white-box-testing

http://www.buzzle.com/editorials/4-10-2005-68350.asp

https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/white-box/259-BSI.html

http://puneetkalra.sulekha.com/blog/post/2007/08/dynamic-white-box-testing.htm

http://www.informatica.uniroma2.it/upload/2007/LSS/03%20-%20blackBox-whiteBox--%20static%20white%20box%20testing.pdf

http://www.astqb.org/educational-resources/syllabi-static3.php

http://msdn.microsoft.com/en-us/library/aa292128(v=vs.71).aspx

http://www.securityninja.co.uk/application-security/a-checklist-approach-to-security-code-reviews-part-2

http://www.mindfiresolutions.com/Code-Review-Checklist-238.php

Unit 5: Special Types of Testing

CONTENTS

Objectives

After studying this unit, you will be able to:

- Explain the need for configuration testing
- Explain the importance of graphical user interface testing

Introduction

Testing a system is meant to ensure an error-free, quality product. A system undergoes several tests before its launch. This chapter emphasizes on configuration testing and graphical user interface testing.

Testing a system is a methodology, which involves both the hardware and the software components of the system. System testing falls under the scope of black box testing. Therefore, system testing does not require a software tester to have a thorough knowledge about the internal design and code of the software. It unravels the defects within the system and between the various links in the code (assemblages).

Under system testing, the system that is to be tested is configured in a controlled environment. Real life scenarios are simulated in the test environment. This type of testing is considered to be complete on two scenarios: First when the actual results and the expected results match, and second when the differences between the actual results and the expected results are well explained and accepted based on client input.

Thus, system testing is considered to be a process of exploring the functionality of the system and identifying the faults within the system. The following example gives you a better idea about **breaking the system.**

Example: Testers input **Name of Country** in a text box that has been designed to accept only **Name of City**. This is done to check the system's response for incorrect data with reference to **breaking the system**.

5.1 Configuration Testing

As defined earlier, configuration testing involves testing the various configuration possibilities for a computer used at home or in an organization.

Configuration testing is the first testing task assigned to a software tester. As a tester you need to ensure that the software works fine without any problem, for all possible hardware combinations.

Testing the system would be a very simple task, when the hardware combinations of computers are identical. In addition, there would be no confusion with the option buttons available to click, and components would interface perfectly every time you use the system.

Did you know? The cost of performing configuration testing is very low, but the benefits are large due to the repeated tests. Therefore, configuration testing is always considered as a cost effective method.

5.1.1 Overview of Configuration Testing

As defined earlier, configuration testing involves testing the various configuration possibilities for a computer used at home or in an organization.

When you are in need of a computer, you would visit a computer showroom or an online store to check the system requirements. You would then decide on a processor, a 32-bit monitor, and so on. Therefore, a simpler definition of configuration testing would be checking the software's functioning ability with various hardware configurations.

Did you know? Configuration testers test printers, Network Interface Cards (NICs), and so on. Configuration testing is also known as portable testing or hardware compatibility testing.

Let us now study the various possible hardware configuration elements that you may have to test.

1. ***Manufacturers:*** Computer manufacturers either design their own computers or obtain certain components from a third party manufacturer to build a computer. Some people also assemble their computers using off-the-shelf components available in the market.
2. ***Components:*** Computers are made up of various components such as system boards, component cards, network cards, disk drives, CD-ROM and DVD drives, video cards, sound cards, input/output cards, and much more specialized hardware for advanced use.
3. ***Peripherals:*** Peripherals are the external hardware devices such as printers, scanners, mouse devices, keyboards, monitors, fax modems, cameras, and joysticks that are plugged into the computer.
4. ***Interfaces:*** Interfaces are the components and peripherals that are plugged into a computer through various internal and external connectors.

Example: Industry Standard Architecture (ISA), Peripheral Component Interconnect (PCI), Universal Serial Bus (USB), Registered Jack- 11 (RJ-11), Registered Jack-445 (RJ-45), and Fire wire are examples of interfaces.

5. ***Options and Memory:*** Components and peripherals with various available hardware options and memory sizes can be bought today. You also have the privilege of upgrading printers to support extra memory and speed up the printing process.

Example: Graphic cards with large memory are used to obtain high resolution and more colors.

6. ***Device Drivers:*** Device drivers are the drivers that help in establishing communication between hardware components and software applications. The device drivers are provided by the hardware manufacturers and are installed in the computer.

Did you know? Device drivers are the software used for testing purposes and are a part of the hardware configuration.

Isolating Configuration Bugs

Isolating the configuration bugs is essential while performing configuration testing. A software tester must be aware of the common bugs that can arise in the process of testing. Configuration bugs are found by performing similar kind of operations on different computers with different hardware setups.

Discovery of configuration bugs can cost you a lot and hence you must make efforts to detect them during the early stages of testing.

Did you know? While playing a game, if you happen to encounter inconsistency in colors or if pixels get stuck, then you have discovered a **display adapter configuration bug**.

Sizing up the Job

The job of a configuration tester is very challenging. The number of software testers required for testing each of the tasks is first addressed. Consider testing a gaming software application on Microsoft Windows operating system. In this scenario, it is necessary to check for appropriate sound effects. So, configuration testing is carried out with the various sound cards and graphics cards, along with the modem specifications.

Did you know? Suppose there are 330 display cards, 240 sound cards, 1,500 modems, and 1,200 printers. The number of test combinations would be 330x240x1, 500x1, 200, which is a very huge number. As a software tester, by using equivalence partitioning you can test the above combination in an effective way.

5.1.2 Identifying Software Configuration

Configuration testing ensures that the software works perfectly fine with the various hardware configurations. Only the most important features that are different from one another are tested using equivalence partitioning. Let us now consider the following example for equivalence partitioning.


Notes

Equivalence Partition

Equivalence partitioning, also called as equivalence classing, is a process of classifying the test cases and grouping them into different classes. This method helps to reduce test cases to a finite number of test cases without compromising on the quality of the test being carried out.

Consider a text box that accepts numeric values ranging from 18 and 60 (18 and 60 are part of the equivalence class).

Valid Classes:

Values from 18 to 60

Invalid Classes:

Values < 18

Values >60

Values such as 19, 24, and 59 fall under the valid class, while values like 17 and 75 fall under the invalid class.

While testing a word pad, you will not test the file save and load features in the configuration. Instead, you will test to create a document that contains various fonts, sizes, colors, pictures, etc. This document is then printed on a selected printer configuration to check for configuration testing.

Design the Test Cases to Run on Each Configuration

To perform configuration testing, a tester must have a good understanding of the product. Black box testing provides better knowledge about the product to the tester. It thereby enables the tester to learn about the features of the product. Hence, test cases are written before configuration testing. Let us now learn some of the steps required to test each configuration.

1. Select the object to be tested.
2. Set up the test configuration from the list.
3. Start the software.
4. Load in the file configtest.doc.
5. Ensure that the displayed file is correct.
6. Print the document.
7. Confirm that there are no error messages and that the printed document matches the standard document.
8. Log any discrepancy as a bug.

This example is just a simple one. However, in reality, the steps involved are detailed, and they specify what to do and what to look for. In addition, the steps followed are simple to run and understand. The tests are iterated until the results are satisfactory.

5.1.3 Deciding the Hardware Configuration

A fair idea about the products and their manufacturers enables the testers to decide the hardware configuration. Based on the hardware configuration, the equivalence partitions are designed and the standards to be followed are identified

The Apple website provides you with information on how to develop and test hardware devices for Apple computers. The Apple website also includes the links pertaining to test-labs with information on conducting configuration testing.

Example: More information about Microsoft's set of standards for both hardware and software can be obtained from the following link.

http://social.msdn.microsoft.com/Search/en-us?query=standards

http://msdn.microsoft.com/en-us/windows/hardware/gg463010.aspx

Configuration Testing other Hardware

Configuration testing involves checking the configurations of the various software and hardware that support a system. Apart from knowing the type of hardware, you also need to know the memory size, CPU speed, and so on and ensure that these hardware types connect to the correct piece of hardware and software configuration.

Notes

Some of the common questions that one needs to answer before performing configuration testing are as follows:

What external hardware will operate with the software?

1. What models and versions of the hardware are available?
2. What features or options does the hardware support?

While performing configuration testing, one must ensure to have relevant set of questions to learn about the system to be tested. Equivalence class partitions of the hardware are created based on input obtained from the people who work with the project manager. Based on their input, test cases are developed and they are run on the appropriate hardware to be tested.

5.2 Graphical User Interface Testing

The Graphical User Interface (GUI) is the front-end that acts as an interface for the users of computer or an electronic gadget.

A software product is developed and designed for use by its customers. Be it office, home or college, you can find several software products used on a regular basis. Therefore, it becomes necessary to check the usability of the software products.

Usability of a software product can be seen in the appropriate functional and effective interaction of the software product. It can be said that usability is very similar to Ergonomics. Ergonomics relates to the science of designing things in such a way that makes the product easy to use it.

As a software tester, you would be the first person to use the software product. Usability of one product differs from that of another. The medium through which you interact with the software is known as User Interface or UI. There are some peculiar cases, one of which is discussed in the following example.

Example: There is no user interface to control fuel or air ratio in an engine. However, extra pressure is required by the gas pedal. The noise from the tailpipe or emission pipe acts as a user interface in this case.

Did you know? Initially, computers had toggle switches and light bulbs as user interfaces.
In the 1960s and 1970s, punch cards, teletypes, and paper tapes were the popular user interfaces.

The GUI has become a de facto standard for user interface in most of the modern technologies.

Some of the reasons for the popularity of the GUI are:

1. It is easy to understand visual interface.
2. It is flexible to use in most of the application areas.
3. It is helpful for people who have difficulty in typing.
4. It provides visibility of multiple windows, to handle information in a better way.
5. It helps in controlling the screens as per user's choice.
6. It facilitates exchange of information because of integration of the packaged and customized applications.

Although GUI has simplified things for users, it has complicated them for a developer. GUI testing can be performed either manually or automatically. Manual testing is a time consuming process, which is performed without the help of automated tools like winruuner, silk test, and Quick Test Professional (QTP). Automated testing is performed by using automated tools like load runner, winruuner and Quick Test Professional (QTP).

Did you know? A Windows based application testing can be classified into Standardization Testing, GUI Testing, Validation Testing, and Functionality Testing.

GUI testing is commonly known as usability testing or user interface testing. Usability testing is the process of checking a product's compatibility, when in use. After completion of the testing process, the software product is released to a set of users as a beta version or a pre-release version. The users evaluate and assess the performance of the software based on customer experience.

GUI testing comprises four stages, the details of which are shown in Table 5.1.

1. Low level
2. Application
3. Integration
4. Non-functional

Table 5.1 GUI Testing Stages

Stages	Testing Involved
Low Level Stage	Checklist testing Navigation
Application	Equivalence partitioning Boundary values Decision tables State transition testing
Integration	Desktop integration Communication Synchronization
Non-Functional	Soak testing Compatibility testing Platform/environment

Source: http://www.comparesuite.com/solutions/tests-automation/hb_gui_testing_introduction.htm

Let us now learn some of the general guidelines for GUI testing. It should be ensured that:

1. All the dialog boxes have a consistent appearance throughout the application system.

Example: If a heading in a dialog box is grey, the headings in other dialog boxes should also be of the same color.

2. Every field on the screen has an associated label.
3. Every screen has an equivalent **OK** and **Cancel** button (with an appealing color combination).
4. Every field in the dialog box supports short cut key functioning.
5. Tab order is set horizontal to the fields, as sometimes it can be vertical.
6. Mandatory fields are marked with red asterisk (*) to denote that they are mandatory.
7. Default key <Enter> is set as **OK** for the dialog box.
8. Default key <Esc> is set as **Cancel** for the dialog box.

Notes

You should remember the following ten points while performing GUI testing:

1. Test for user friendly labels, messages, related message content, and understandability of the message.
2. Test ease of navigation.
3. Test availability of help for a particular operation.
4. Test use of colors, fonts, alignment, and tab orders.
5. Test end to end navigation.
6. Test functionality of control objects like buttons, textbox, list box, and so on.
7. Test for the risk of critical defects.
8. Test the strict adherence to time and schedule.
9. Test the risk of Quality Assurance (QA) resources yet to be familiarized on the new application.
10. Test the risk of low priority aspects of the application that is not being tested till the later stages.

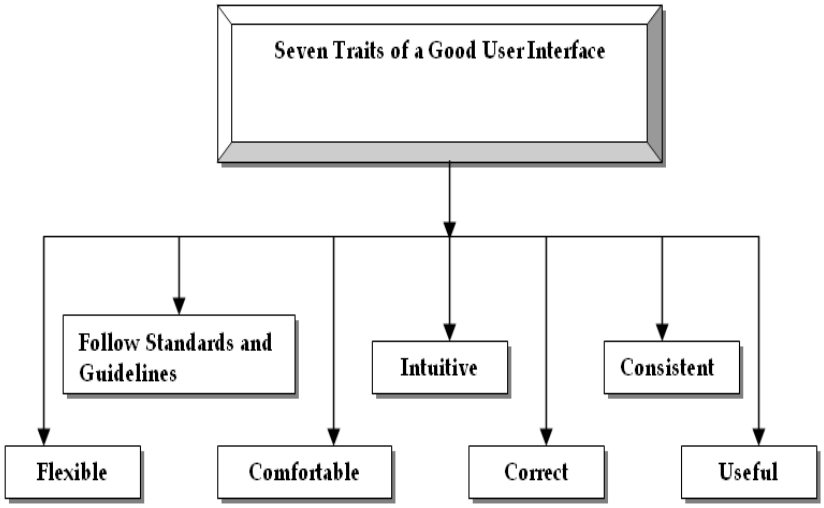
5.2.1 Standards and Guidelines

Standards and guidelines are necessary to obtain a quality software product with good user interface and reliability.

A software tester needs to pretend to be the user while testing for usability and locate the possible errors or problem-prone areas in the software product.

The seven important traits of a good user interface are as shown in figure 5.1.

Figure 5.1: Seven Good User Interface Traits



We will now discuss the seven traits of a good user interface.

1. ***Follow Standards and Guidelines:*** It is very important for every software product to adhere to standards and guidelines. When the software is running on Windows platform, it is assumed that it is adhering to a set of standards.

Did you know? The standards and guidelines for Apple can be obtained from the book titled Macintosh Human Interface Guidelines published by Addison-Wesley.

While testing for usability of a software product on a specific platform, you need to adhere to the standards and guidelines of that platform. Test cases should be created based on the standards and guidelines generally developed by the usability testing experts.

At times, the software product you are testing might not have a standard. In such situations, the design team creates a usability standard for the software product.

Notes

Standards and guidelines provide a better idea about when to use the check boxes, the option buttons, warnings, critical messages, and information especially in ambiguous situations.

2. ***Intuitive:*** The Micro Instrumentation Telemetry Systems (MITS), Altair 8800 was the first personal computer released in 1975. The user interfaces were switches and lights. This computer was created for hobbyists. However, in today's world, a customer looks for more in every software product. In view of the customer demands, we need to ensure that the following points are considered while performing user interface testing.

Some of the things to be considered while performing the user interface testing are:

(a) Check if the user interface is clean, unobtrusive, and not cluttered with options and information. The user interface must not get in the way of what you want to perform or the functions you need, and the expected response must be obvious.

(b) Check if the user interface is organized and laid out well. Ensure that it allows you to navigate from one function to another. At any given time, you should be able to do nothing, back up, or back out.

(c) Check for excessive functionality. Ensure that the software does not try to do too much, either as whole or as a part.

(d) Check if the **help system** really helps when everything fails.

3. ***Consistent:*** Consistency within the software and with other software is a key att**ribute**. Inconsistencies when moving from one program to another frustrate the users. Thus, it is necessary to follow a standard for the software or the platform, else attention must be paid to the features of the software to ensure that similar operations are performed in the same way.

The consistency criteria to be outlined while performing usability testing are:

(a) ***Shortcut Keys and Menu Selections:*** The shortcut keys are similar to accessing **Help** by pressing **F1** in Windows.

(b) ***Terminology and Naming:*** You need to look for same terms used throughout different versions of the software and check whether the features are named consistently.

Example: Is the **Find** always known as **Find** or by any other name like **Search**?

(c) ***Audience:*** You need to check whether the software consistently addresses all kinds of audience level.

(d) ***OK and Cancel Button Locations:*** You need to check whether the location of the buttons remains same from one platform to another.

Example: The keyboard equivalents present on screen must be consistent. The **Escape (*Esc*)** key should always cancel operations.

4. ***Flexible:*** Flexibility relates to the ease with which the user performs tasks as per requirements.

Example: A calculator having both scientific and standard normal view.

Flexibility in software provides the following features:

(a) ***State Jumping:*** Software which is very flexible, gives more options to accomplish the same task.

(b) ***State Termination and Skipping:*** Software with power-user modes allows the user to skip numerous prompts or windows and go to the destination directly.

Example: A voicemail system in a hospital which allows you to directly punch in the extension you require.

(c) ***Data Input and Output:*** Users demand ways in which they can enter data and view their results.

Example: It is possible to enter text on a WordPad in six different ways which include:

Type texts, paste text, load text, insert an object, and drag it using the mouse from another program.

5. ***Comfortable:*** The user must be comfortable using the software.

(a) ***Appropriateness:*** The software designed must have a proper look, feel, and relate to what it is supposed to do.

(b) ***Error Handling:*** Programs must be written in such a way that they warn users before a critical operation and also allow the users to restore the lost data.

Example: The **Undo** feature

(c) ***Performance:*** Performance does not refer to speed, instead it implies that more than one program can flash error messages at a greater speed.

Example: The **Status** bars that display the accomplishment of a task.

6. ***Correct:*** Testing for correctness implies checking whether the user interface accomplishes what it is supposed to do. The appearance of the **hourglass** symbol indicates that the software is busy and cannot accept any input at that particular time.
7. While testing, a tester needs to pay attention to some areas like:

   (a) ***Marketing Differences:*** Usually, the software will have some marketing material also. Check whether the software performs operations as mentioned in the marketing material. You must also ensure that the software is compared to the sales information and not the specification.

   (b) ***Language and Spelling:*** Sometimes errors are created due to the poor language and vocabulary of programmers and writers. Thus, messages like '**If there are any discrepancies, please contact us immediately to ensure timely delivery of the products that you ordered'** may appear.

   (c) ***Bad Media:*** Media is considered as the channel through which all the supporting items such as icons, images, sounds, and videos go with the software user interface. Thus it is essential to check that none of the supplied media are in bad condition.

Example: Sound should be of the same format and sampling rate, as specified for the application.

   (d) ***What You See Is What You Get (WYSIWYG):*** Always check whether the user interface displayed is the one you have.

Example: When printing a text, ensure that the previewed text is printed exactly same as it is displayed in print preview.

8. ***Useful:*** A user interface has to be useful. The features of the product must be easy to use.

5.2.2 Accessibility Testing

Accessibility testing is a technique used to ensure that the software product is accessible to people with disabilities such as visually challenged, physically challenged, and hearing defects. Accessibility testing is also known as testing for the disabled.

Did you know? A US government survey, called as Survey of Income and Program Participation (SIPP), conducted during the year 1997 revealed that 53 million (20% of the population) in the country had some form of disability. Thus there is greater need to test for this aspect.

Accessibility testing is classified into four groups based on types of accessing difficulties and issues. They are:

1. ***Visual Impairments:*** People with visual impairments like blindness, restricted vision, color blindness, and so on can have the ease of working with the software products. Generally people with this kind of disability (visual impairments) make use of the assistive technology software. Assistive technology through screen reading software helps the blind read the content. This technology simulates the human voice reading the text on computer screen or renders hard copy output into Braille. An example of such technology is Job Access With Speech (JAWS).
2. ***Motor Skills:*** Motor skills involve the skills pertaining to usage of keyboards and mouse. People with an inability to use the keyboard or mouse and to make fine movements on the software product come under this category.
3. ***Hearing Impairments:*** Hearing impairments relate to people with a reduced or total loss of hearing.
4. ***Cognitive Abilities:*** The cognitive abilities relate to reasoning and judging skills of an individual. People with reading disabilities, dyslexia, or memory loss fall under this category.

While performing accessibility testing, a tester must ensure that an accessibility checklist is used to certify the compliance related to accessibility. The checklist provides information about what needs to be tested, how it should be tested, and the status of the software product with respect to various access related problems.

For a successful accessibility testing, the test team follows a separate cycle. The management also provides the test team with the required tools for conducting accessibility testing.

Following is a simple test case for accessibility testing:

1. Ensure that all the functions are available with the keyboard, without having to use the mouse.
2. Ensure that information is visible when the display settings are set to high contrast modes.
3. Ensure that screen reading tools read the available texts as pictures, and images contain alternate texts associated with them.
4. Ensure that the product-defined keyboard actions do not affect keyboard shortcuts associated with accessibility.


Task

Assume that you have been given the responsibility to test a modem. Write test cases for a modem based on that.

Accessibility testing requires more than one tool to test. Two categories of tools are used which include:

(a) ***Inspectors or Web Checkers:*** This category enables the developer and the tester to know the exact information provided to an assistive technology. Tools like inspect object are used to obtain information given to a system on the assistive technology.

(b) ***Assistive Technologies (AT):*** This category is used by people with disability. Tools like screen readers, screen magnifiers, and so on are used to enhance accessibility. Testing an assistive technology is performed manually to have a better understanding of the interaction between AT and the product and documentation.


Caution

When using a screen reader, ensure to include tests for all the actions performed by the user, such as installing and un-installing the product.

If a function cannot be performed using an AT, it may be considered accessible if it has a command line interface to perform that function.

Thus, we can conclude that special types of testing are required for a product to be launched in the market. The process of testing is carried out from the customer's point of view. Today, there is a greater requirement for applications based on user requirements. As per the Survey of Income and Program

Participation (SIPP) conducted in the United States, the demand for accessibility testing for the newly developed features has come into existence.

Challenge of Access

Access' is one of the leading information technology companies specialized in the development of GUI components for mobile phones built on Symbian, MS Smart phone, J2ME, and other platforms.

The company manufactures the latest technology bio-medical equipment. This provided the company an advantage over the other existing companies. However, the user interfaces for some of the medical equipments needed the language generating graphic shell environment with emphasis on the examination of shell performance.

Solution

The situation was thoroughly analyzed and the Access' team decided to change the simulator by adding the analysis fields of states and event processing. This gave them the option of executing every language command at least once. Thus, the displays for all the state changes were checked.

At first, the team at Access' checked the shell on all the devices and simulated and compared the real results with the expected ones. Later they analyzed the types of testing to be carried out and developed test cases accordingly. These developed test cases were tested for testing the real devices and then the tests (scripts) were collated.

Results

The use of automated testing tools made the process of testing efficient. Discrepancies were identified and the relevant results to the test cases were inserted. It was possible to test at a faster pace with the simulator modification, thus reducing the time consumed for testing. New functionality and error fixing became transparent. It also provided the result tracing feature. Thus, Access' saw a great benefit in getting a self testing environment.

Question

1. Explain how Access' managed to achieve efficient testing?

Adapted from http://www.bughuntress.com/portfolio/case-studies/user-interface.html

5.3 Summary

- Special types of testing provide a better understanding about configuration testing and the graphical user interface testing used for testing the system before release.
- The Special types of testing are the tasks assigned generally to new software testers, to introduce them to the equivalence partitioning skills.
- System testing provides opportunity to the software testers to work with the project team members, which is an overwhelming experience for a new tester.
- Configuration testing is a testing task that ensures that the software works fine without any problem, for all possible hardware combinations.
- Configuration testing is based on manufacturers, components, peripherals, interfaces.
- A software tester is the first person to check the usability of the software product before its launch.
- Testing the Graphical User Interfaces can be vague or endless, but when they do not meet the criteria, a bug is supposed to have occurred.

5.4 Keywords

Assemblages: *A* system made by putting in objects together.

De Facto Standard: A standard which has been accepted and adopted but not been defined and endorsed by the standards organization.

Hobbyists: Someone who enjoys doing something and is not concerned with the intricacies of the product he/she is working with.

Network Interface Cards (NIC): A network interface card (NIC) is a computer card that is installed in a computer to connect it to a network.

Quick Test Professional (QTP): QTP is a popular test automation tool mainly for functional testing.

Silk Test: SilkTest® is an automation tool used for regression testing, delivering and advanced test automation capabilities.

Unobtrusive: Not noticeable.

5.5 Self Assessment

1. State whether the following statements are true or false:
   (a) Real life scenarios are simulated in the test environment and are tested on the system as required in real life.
   (b) Configuration testing deals with the process of testing the system with both software and hardware configurations.
   (c) Configuration testing is also known as portable testing.
   (d) The GUI (Graphical User Interface) is the front-end that acts as an interface to the users.
2. Fill in the blanks:
   (a) System testing unravels the defects within the system and between the_____.
   (b) The device drivers are provided by the _____ and are installed in the computer.
   (c) Equivalence class partitions of the hardware are created based on input obtained from the people who work with the _____.
   (d) Usability is something very similar to_____.
3. Select a suitable choice for every question:
   (a) Identify which of the following teams performs the system testing.
      (i) Test team
      (ii) Development team
      (iii) System analysis team
      (iv) Management team
   (b) The cost of performing configuration testing is very less due to
      (i) System testing
      (ii) Compatibility testing
      (iii) Regression testing
      (iv) Usability testing
   (c) Identify which of the following is an interface.
      (i) Printer (ii) RJ-45 (iii) CD-ROM (iv) DVD burner

(d) Based on which of the following criteria are the most important programs decided for testing?

(i) Popularity (ii) Time (iii) Criticality (iv) Software

5.6 Review Questions

1. Justify when a bug is considered to be due to configuration related problems.
2. Do you believe that Configuration testing has to encompass all components manufactured by third party manufacturer? Why so?
3. Do you believe that equivalence partitioning the hardware brings efficiency in configuration testing? Justify.
4. Do you believe that GUI testing can enhance the product's usability? What are the different levels in which GUI can be carried out?
5. Assume that you have been assigned the task of testing a web based application. What would be the main traits you look forward while testing the GUI?
6. Can a software product be released with a configuration bug? Explain.
7. "The GUI has become a de facto standard for user interface in most of the modern technologies." How would you justify this?
8. "Configuration testing ensures that the software works perfectly fine with the various hardware configurations. " How would you guarantee the same?
9. Does testing takes care of specialized software developed for specially challenged people (people with disabilities)? How, in your opinion, is it necessary in today's context?
10. When do you think that a software product is designed badly? List out the possible user interface errors in the software product.

Answers: Self Assessment

1.	(a) True	(b) True	(c) True	(d) True
2.	(a) Assemblages	(b) Hardware manufacturer	(c) Project manager	(d) Ergonomics
3.	(a) Test team	(b) Regression testing	(c) RJ-45	(d) Popularity

5.7 Further Readings


Books

Patton R (2006), Software Testing-Second Edition, USA, SAMS Publishing
Hutcheson, Marnie L (2003), Software Testing Fundamentals, USA, Wiley Publishing


Online link

http://www.robdavispe.com/free2/software-qa-testing-test-tester-2047.html
http://www.ercim.eu/cyclades/vip/del/D4.2.1.pdf
http://www.businessdictionary.com/definition/system-testing.html
http://www.nsc.liu.se/~boein/f77to90/a4.html
http://www.communitymx.com/content/article.cfm?cid=8897D

Unit 6: Compatibility Testing

CONTENTS

Objectives

After studying this unit, you will be able to:

- Describe compatibility testing
- Describe backward and forward compatibility
- Describe the standards and guidelines

Introduction

Any software product can be architecturally well-designed, perfectly coded, but unless it clears the aspect of being able to function across the myriad computer system of target users, it will never be able to fare well in the market.

Compatibility testing gives the developers the confidence of the application's compatibility with the computing environment. Compatibility testing gives clarity of the application's ability to coexist with other functions and how well it gels with other systems. It can be categorized as a software non-functional test. Today, Compatibility testing is mandatorily carried out to all applications, since it is used to detail the specification of the product, namely the different types of system hardware and software that works with the system.

6.1 Compatibility Testing

Compatibility testing relates to testing the interactions between two different software, to make sure that both the software work correctly. The need for compatibility testing is high today, because most of the consumers demand data-sharing options with different types of software programs from various vendors.

Earlier, most programs were developed as standalone applications and ran only in a known environment setup. The reason was that the developers hesitated running the program on a different environment fearing corruption of the program. However, today there is a need for most programs to be compatible with different operating systems and Web browsers. In addition, these programs also

need to constantly import and export data to other programs that run simultaneously on the same hardware.

As a software compatibility test engineer, you need to ensure that the interaction between different software operates or functions as required by the users. Issues pertaining to the way in which the software functions with the various operating systems and the different types of hardware and software systems are identified.

Did you know? Compatibility testing helps you to avoid the dangerous and expensive hazards or troubles that can occur after the product is released into market.

6.1.1 Overview of Compatibility Testing

Compatibility testing also relates to testing the interactions between programs or software either in the same computer or between different computers that are located thousands of miles away connected through the Internet.

These interactions can be as simple as saving the data to a Compact Disc (CD) and carrying it to another computer situated across the room. Let us look at some of the examples for compatibility testing.

Example:

1. Copying text from a web page and pasting it on to a document in word processor.
2. Saving data related to accounts from one spreadsheet program to another spreadsheet program.

Compatibility testing is carried out using real-time environments and not virtual environments.

Testing the compatibility of the product varies from one testing team to another, since each testing team will be assigned specific tasks to test. These tasks differ based on the system requirement and the software on which it runs.

The software for standalone medical devices run on their own operating systems, store data on their own memory, and do not connect to any other device. Hence, in this scenario, there is no room for considering compatibility.

Did you know? Today, some companies outsource compatibility testing to third parties. One such example is ApTest.

ApTest is an expert in testing product compatibility with both hardware and software environments.

Notes

If you have been assigned the task of conducting compatibility testing on a piece of software, the following checklist must be followed:

1. What are the various platforms and application software your software is designed to be compatible with?
2. What are the compatibility standards and guidelines to be followed and how should your software interact with other software?
3. What are the types of data that your software will use to interact and share with other software and platforms?

Platform and Application Versions

The task of the marketing team is to select the target platform and compatible applications because the operating system, Web browser, compatible versions, and other features are designed based on customer perspective. The platform and application versions enable the development and testing teams to decide on what is to be done.

Example: On most of the software packages or start up screens you may come across the following:

Works best with

Requires Windows XP or greater

For use with Unix or Linux 2.6.10 only

Compatibility testing involves checking the compatibility of an application or website with several browsers, operating systems, and hardware. This testing is conducted on an existing environment either manually or on an automated basis.

Through compatibility testing, one can be sure that:

1. The software is rigorously tested with all the operating systems, software applications, and hardware.
2. The code is stable in all environments and the error messages or user interactions are handled and presented in the same way regardless of the operating system.
3. The manual tests are carried out by assigning different team members to work on different screen resolutions to check for issues that arise.
4. The varying connection speed available to the user base is taken into account.
5. The checks on the non-functional test teams are planned while running the tests.
6. The performance of a system, application, or website on a network with varying parameters like bandwidth, variance in capacity, and operating speed of the hardware are evaluated.
7. The system or application performance in connection with the various systems or peripheral devices is evaluated.
8. The application or system performance with respect to the database it interacts with is evaluated.

6.1.2 Backward and Forward Compatibility

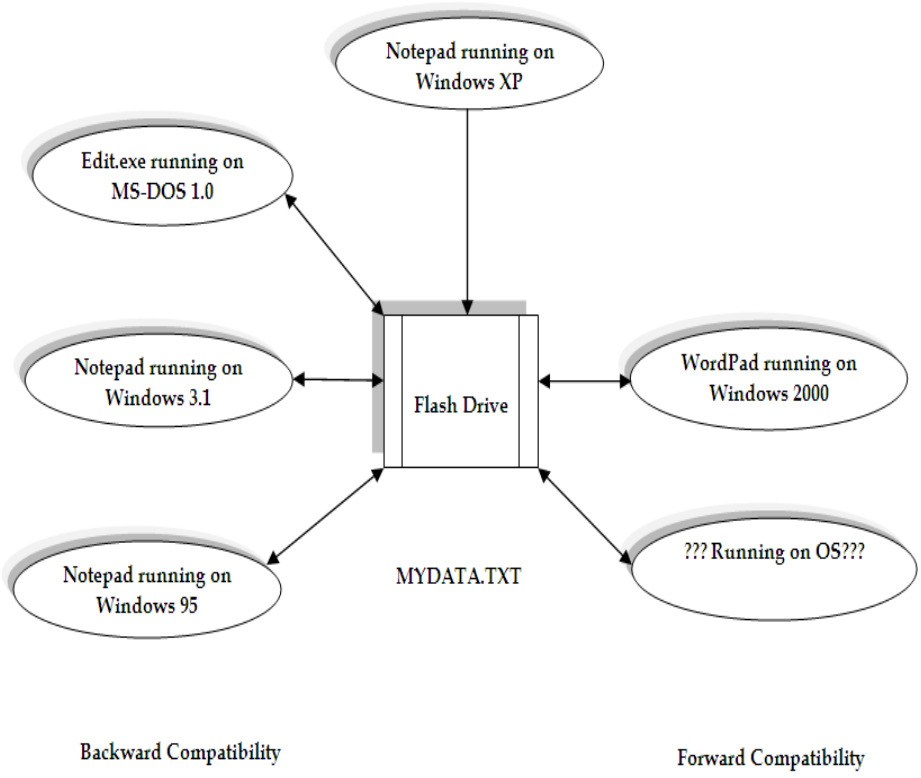
The two terms associated with compatibility testing are backward compatibility and forward compatibility. Backward compatibility means that software works fine with the previous versions of the software. Forward compatibility implies that the particular software works well with the future versions of the software.

Example: Backward compatibility: The compatibility of Microsoft Word 2007 version with Microsoft Windows XP operating system.

Forward compatibility: The compatibility of Microsoft Word 2003 version with Microsoft Windows 7 operating system.

We will now analyze backward and forward compatibility through a simple example:

Figure 6.1: Backward and Forward Compatibility



The simplest example for backward and forward compatibility is the .txt or text file. Figure 6.1 shows a text file created using Notepad 98 that runs under Windows 98 (it is backward compatible) and can be tested all the way back to MS-DOS 1.0. It is also forward compatible to Windows XP Service Pack 2 and is expected to go beyond that as well.

It is not necessary for all files and software to be backward and forward compatible. Software designers make decisions related to testing requirements for forward and backward compatibility of the software. Backward compatibility is similar to the concept of using old programs once again with new standards.

Forward compatibility is the compatibility checking of a product with the future versions of the software. For example, the new changes that have been considered for the next FORTRAN version are improved with parallel treatments like interrupt handling, parameterized data types, and inherited data types.

Example:

1. When the migration occurred from Fortran 66 to Fortran 77, the extended DO-loop was removed considering the fact that the extended DO-loop means that if you do not change any of the DO-loop parameters you can jump out of the loop and then jump in again. This is similar to the concept of structured programming.
2. The Hollerith constants were removed (except in FORMAT).

The above example implies that some programs that work with Fortran 66 do not work with Fortran 77. Hence, manufacturers have included these two concepts in their FORTRAN implementations.

The incompatibility between Fortran 66 and Fortran 77 is related to the assumed size allocation of dummy arrays.

The 'obsolescence concept' emphasizes on some of the constructs that may be removed in the next change of FORTRAN.

Some of the constructs that have been treated as obsolescent are:

1. Arithmetic IF-statement
2. Control variables in a DO-loop that are floating point or double-precision floating-point
3. Terminating several DO-loops on the same statement
4. Terminating the DO-loop in some other way than with CONTINUE or END DO
5. Alternate return
6. Jump to END IF from an outer block
7. PAUSE
8. ASSIGN, assigned GOTO, and assigned FORMAT, which relate to the whole statement number variable concept
9. Hollerith editing in FORMAT

Information on obsolescence can be obtained from Status of FORTRAN 95, which provides details pertaining to suggestions for the next standard, along with the new list of deleted features and the revised list of obsolescent features.

Parallel Extensions

A group of High Performance FORTRAN Forum worked on developing the extension to Fortran 90 with parallel extensions. The intention of this project was to provide a portable language that could be efficiently used on different parallel systems. This project was ready by May 1993, with a de facto standard.

6.1.3 Testing Multiple Versions

To test various versions of platforms and software applications is a challenging task. We will now consider a situation where a compatibility test is to be done on a popular operating system. The programmers have fixed several bugs and have also improved performances by adding new features to the existing code. There are thousands of existing programs for the present version of the operating system. The ultimate aim of the project is to ensure 100% compatibility. Equivalence partitioning is appropriately applied to reduce the job of testers.

The task of compatibility testing starts with the equivalence partitioning of all possible combinations of the software. This is done to ensure that the equivalence sets verify the accuracy of the interaction between the software. Although one can test all the possible software programs on the operating system, only the most important ones are finalized and tested.

The criteria for finalizing the most important programs are as follows:

1. ***Popularity:*** Select the first 100 or 1000 popular programs based on sales data.
2. ***Age:*** Select programs and versions that are less than three years.
3. ***Type:*** Select software from every relevant category by segregating the applications into types like accounting, databases, and communications.
4. ***Manufacturer:*** Select software based on the company that has created it.

The platform versions and the software applications with which the software is to be tested must be decided before performing a compatibility testing.

Let us now learn how to test websites against multiple browsers and multiple browser versions. Although, Internet Explorer 9 (IE9) emulates the older version of Internet Explorer (IE), the emulations are not always accurate. This is because developers require simple, convenient ways to run multiple

versions of Internet Explorer on one computer. The Windows XP mode of the Windows 7 operating system is used as the best mode for testing websites across versions of IE on one computer.

Running Multiple Versions of IE Using Windows XP Mode

One way to run multiple versions of IE on a computer is to run the older version/s of IE using Windows XP mode on a computer running Windows 7 Professional, Enterprise, or Ultimate edition. (Windows XP mode is an optional downloaded feature of Windows 7 Professional, Enterprise, and Ultimate editions that provides you a pre-installed image of Windows XP SP3 that you run using Windows Virtual system. A recently released update allows Windows XP Mode to run on a CPU without hardware virtualization.)

Did you know? It is also possible to set up and run multiple Windows XP modes on a Windows 7 machine, thereby allowing you to run IE versions 6, 7, 8, and 9 simultaneously on a single machine.

The Microsoft-sanctioned way of testing multiple versions of IE is performed by installing the Microsoft Virtual PC to its software program. The Microsoft Virtual PC was earlier a commercial software, but now available free of cost. The Microsoft website does the job of summing up the Virtual PC.

The **Virtual PC 2004** facilitates the creation of separate virtual machines on the Windows desktop. Each of these virtual machines virtualizes the hardware of a complete physical computer. The Virtual PC helps customers install any number of operating systems on a virtual machine.

The Virtual PC helps in setting up multiple operating systems on multiple virtual computers, where each virtual computer has its own version of IE. This provides a clear picture on how a physically separate machine can render a website in each version of IE. The Virtual PC allows us to customize each browser by installing the flash player plug-in, thereby giving a broader perspective for test scenarios. You cannot simply install the copy of Windows XP in each virtual machine. If you require three virtual computers running on Windows XP, then you need four licenses for Windows XP -- one for your own computer, while the rest of them are for the virtual ones.

There are certain disadvantages of using the Microsoft Virtual PC:

1. You need a separate license for each of the Windows operating systems of the virtual machine created. (Today, large organizations have no trouble in obtaining extra licenses to install virtual machines. You can also consider buying the older versions of Windows like Windows 2000 to install in a virtual computer).
2. You cannot access the browsers that are installed in virtual machines from Dream Weaver's Preview in browser command.
3. Running several operating systems on one computer is resource-intensive, because the computers require more power and memory to handle the load.

Example: VMware's free VMware Player, and VMware Server software work similar to Virtual PC and have the same pros and cons.

6.1.4 Standards and Guidelines

It is important to have certain standards and guidelines because standards and guidelines enable a software tester to follow the right approach of testing.

There are two levels of standard requirements. They are as follows:

1. ***High-Level Standards and Guidelines:*** High level requirements are the standards that guide a product's general operation, design, and supported features. For high-level standards and guidelines, it is important to analyze whether the software that is to be tested can run on Windows, Mac, and Linux operating systems. It is also important to know on what browsers the software will run. This is done by considering the platform that has its own standards and

guidelines. It is possible to pass the compatibility testing by adhering to these standards and guidelines.

Some of the examples for logo requirements are as follows:

(a) The software should support the mouse device with more than three buttons.

(b) The software should support installation on all the disk drives of a computer.

(c) The software should support longer filenames.

(d) The software must not read, write, or even use the old system files win.ini, system.ini, autoexec.bat or config.sys

Example: The Microsoft logo must pass compatibility testing through an independent testing laboratory. This is to ensure that the software runs stable on an operating system. For more information on logo requirements visit www. msdn.microsoft.com/certification.

Figure 6.2: Microsoft Windows LOGO

Source: http://windows.radified.com/windows_xp_install.htm

Did you know? Although the requirements sound very simple, they sum up to a huge document consisting of over 100 pages.

2. ***Low-Level Standards and Guidelines:*** Low level requirements are standards that examine fundamental details such as file formats and network communication protocols. Both these requirements should be tested to assure compatibility. The low-level standards are considered to be more important than the high-level standards. It is possible to create a program to run on Windows, although it may not have the design of the Windows software. Users will not appreciate the changes but will only emphasize on the use of the product.

Did you know? If the software is a graphic program that saves files to the disk as .pict files (a standard Macintosh file format for graphics), then the program does not follow the standard for .pict files. As a result, users will be unable to view the files on any other program. Therefore, the software will not be compatible with the standard and will be a short lived product.

Low-level standards should never be taken for granted when it comes to communications protocols and programming language syntax. Therefore, low-level standards should adhere to published standards and guidelines. In most cases, the low-level requirement standards are considered as an extension of the software specification.

Let us now consider the following example.

Example: When the software saves and loads its graphics files as .bmp, .jpg and, .gif formats, the standards for these formats and the design tests are checked for adherence.

6.1.5 Data Sharing Compatibility

Data sharing involves the process of sharing the data among various applications. Checking the compatibility of the various applications during data sharing is known as **Data Sharing Compatibility**. Only a well-written program that adheres to standards and guidelines enables the users to transfer data to and from software efficiently and is considered to be a compatible product. The simplest way to transfer data from one program to another program is by saving and loading the data on storage devices.

1. ***File Save and File Load:*** Disk sharing becomes a possibility only when one adheres to the low-level standards for the disk and file formats.

Example: Saving the file and loading the file are the most common data sharing methods.

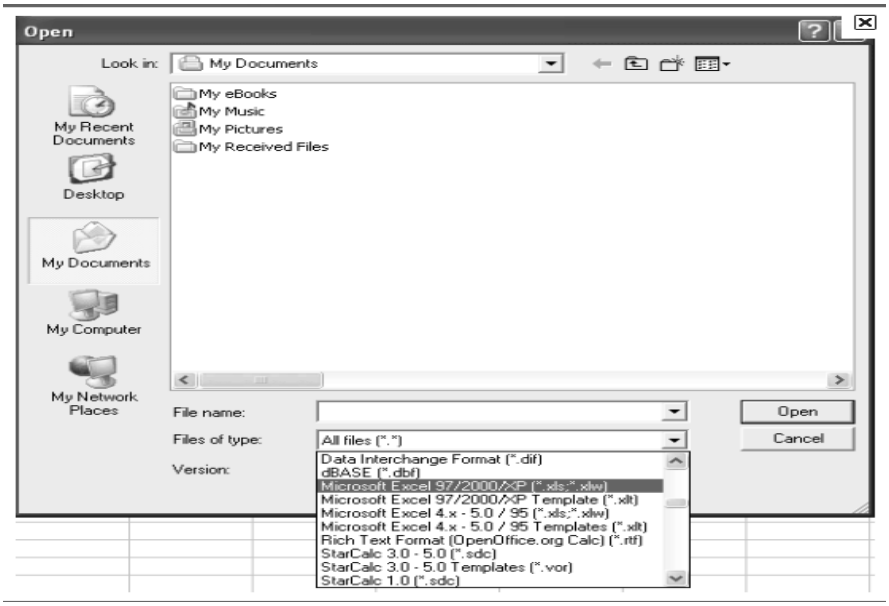
You can copy the data to a floppy disk or hard disk and carry it to another computer and load it to run on different software. You must also ensure that the data format of the files must meet the standards that are compatible on both computers.

2. ***File Export and File Import:*** Many programs are compatible with their previous versions and with other programs through file export and file import.

Did you know? The word processor can import 23 different types of file formats.

You can get a better idea about Microsoft file export and file import from figure 6.3.

Figure 6.3: File Export and File Import

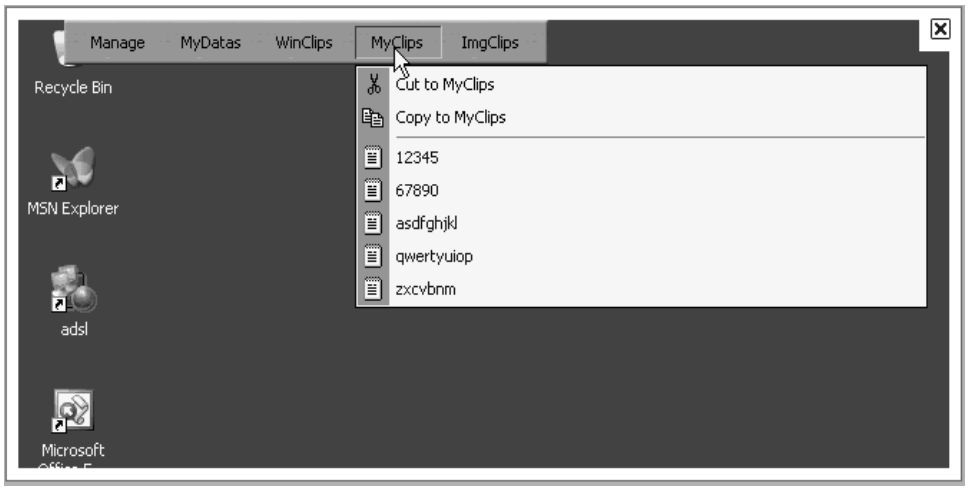


To test the file import feature, you need to create test documents in each compatible file by using the original software that was used to write the format. The documents are then partitioned based

on the equivalence partition method and are then checked whether the importing codes convert to the new format.

3. ***Cut, Copy, and Paste:*** Cut, copy, and paste are the most common methods used for sharing data between programs without having to transfer the data to a disk. The process of transfer takes place through an intermediate program known as clipboard. Figure 6.4 provides a better understanding about data sharing.

Figure 6.4: Data Sharing for Cut, Copy, and Paste



A **clipboard** is designed to hold several data types. The most common ones are texts, pictures, and sounds. The data types can be in different formats -- for example, texts can be plain texts, pictures can be bitmaps, and so on.

Each time a user performs a cut or copy, the selected data is placed on the clipboard. When the paste operation is performed, the data gets copied on the destination software on the clipboard. Some of the applications accept only certain data types or formats to be pasted on them. This is illustrated in the following example.

Example: A paint application accepting only pictures and not any kind of text.

Notes While performing compatibility testing, you must ensure that data is properly copied in and out of the clipboard to other programs.

4. ***DDE, COM, OLE:*** Dynamic Data Exchange (DDE – pronounced as D-D-E), Component Object Model (COM—pronounced as oh-lay) and Object Linking and Embedding (OLE) are the various methods used in Windows to transfer data between two applications The main difference between the clipboard method and the DDE and OLE data method is that the clipboard method allows data flow from one application to another on a real time basis, whereas with the DDE and OLE, data transfers can be achieved automatically.

6.2 Summary

- Compatibility testing is required in every application as there is a great need to check for issues related to compatibility.
- Generally, compatibility testing is conducted on a new operating system to check for its compatibility with word processors or graphics programs.
- Compatibility testing for application programs is conducted on different platforms.
- Equivalence partitioning method is considered to be one of the best choices for compatibility testing.
- The high level and low level standards and guidelines help to have a reliable software product.
- The data flow which occurs between software programs are tested to ensure that one software program is compatible with another software program.
- Compatibility is assured based on how the data exchange happens.

6.3 Keywords

De Facto Standard: A standard which has been accepted and adopted but not been defined and endorsed by the standards organization.

Disk Files: Disk files are file systems which manage to store data on permanent storage devices.

Parallel Processors: Performing several functions at once.

Vector Processors: A computer which has built in instructions and can perform operations on vectors simultaneously.

Virtualization: Creating a virtual replica of something like operating system, memory, and so on.

6.4 Self Assessment

1. State whether the following statements are true or false:
   (a) Compatibility testing is not a concern today.
   (b) Today, the Microsoft Virtual PC is commercial software.
   (c) There are two levels of requirement standards for compatibility testing.
   (d) The high level requirement standards are considered as an extension of the software's specifications.
   (e) The simplest way to transfer data from one program to another program is by saving and loading the disk files, which is known as data sharing.
2. Fill in the blanks:
   (a) The best approach followed by software testers to test effectively is through _____.
   (b) Compatibility testing is carried out using _____environment.
   (c) The concept of using old programs once again with new standard is similar to _____.

3. Select a suitable choice for every question:

   (a) When we say that the software can work well with the previous versions and the present versions, what does this imply?

      (i) Backward compatibility

      (ii) Forward compatibility

      (iii) Hardware compatibility

      (iv) Backward and forward compatibility

   (b) Based on which of the following criteria are the most important programs decided for testing?

      (i) Popularity

      (ii) Time

      (iii) Criticality

      (iv) Software

   (c) Data transfer happens through an intermediate program known as

      (i) Clipboard

      (ii) Data sharing

      (iii) User interface

      (iv) Portable testing

6.5 Review Questions

1. ″Compatibility testing is carried out using real time environments″ Justify.
2. What would your approach be towards the data file formats in the process of compatibility testing? Elaborate.
3. Assume that you have developed a windows based application that has the capacity of Dynamic Data Exchange (DDE) and Object Linking and Embedding (OLE). What kind of compatibility testing would you suggest?
4. Compatibility testing is not mandatory for standalone medical devices. Justify
5. "The need for compatibility testing is high today" Why do you think so?
6. Assume that you have developed the next version of software (Version 2.0) and also a patch for the existing version (Version 1.3.4). What type of compatibility testing will you carry out for these applications? How will you ensure its compatibility?

Answers: Self Assessment

1. (a) False (b) False (c) True (d) False (e) True
2. (a) Equivalence partitioning (b) Real time (c) Backward compatibility
3. (a) Backward compatibility (b) Popularity (c) Clipboard

6.6 Further Readings


Books

Patton R (2006), Software Testing-Second Edition, USA, SAMS Publishing

Hutcheson, Marnie L (2003), Software Testing Fundamentals, USA, Wiley Publishing


Online link

http://nresult.com/testing-services/compatibility-testing.php

http://www.uptodate.com/contents/compatibility-testing

http://productdevelop.blogspot.com/2010/08/overview-and-features-of-compatibility.html

http://www.nsc.liu.se/~boein/f77to90/a4.html

http://www.communitymx.com/content/article.cfm?cid=8897D

Unit 7: Documentation and Security Testing

CONTENTS

Objectives

After studying this unit, you will be able to:

- Explain importance of documentation testing.
- Describe the need for security testing.

Introduction

Testing is an important step in any development process for both software and hardware products. Testing the functionalities of the end product is the main focus of all testing techniques. There are many other facets of testing which brings thoroughness to the product. Documentation testing is an example, which involves testing the accuracy of various kinds of documents which are part of the software application.

The process of software testing remains incomplete if the documentation related to the software is not tested. The tester has to make sure that the documentation elements accompanying the software are error free. The main objective of any tester performing documentation testing is to make sure that the testing meets the following objectives:

1. Whether the information mentioned in the documentation is available in the product.
2. Whether the required information of the product is provided in the documentation.

This unit also deals with security testing that helps to bring the confidence in the product security. Software security is another major area of concern for any organization. This is one of the most important elements that determine the quality of the software product. Any compromise or errors in software security will cause both financial and data losses to the users. The testing process must make sure that any vulnerability in the software is detected and resolved before the software is released to the market.

7.1 Documentation Testing

What is documentation? Documentation refers to written information that defines, describes, specifies, reports or certifies the activities, requirements, procedures or results of the software application. It also includes pictorial information. Documentation is used to provide information about the product such as design documents, code commands, white papers, and so on. It refers to the product's technical manuals which are made available both online and in the form of a printed book.

Example: When you purchase a mobile phone there is a manual present along with the mobile phone. This manual will provide all the necessary information about the mobile product such as model number, list of features, information about the keys, safety tips, and so on. This 'documentation' helps the user to know and understand the features easily.

Notes

The end user can be a common man who might not understand the technology and just uses the application. Alternatively, the end user can be a highly skilled technician, who will install or repair the system. Therefore, the type of documentation and information covered varies depending on the end users.

Documentation meets its objective only if it provides necessary and complete information to the end users or customers. Therefore, it is very important to make sure that no error or incorrect information is included in the documentation. In order to remove such errors the process of documentation testing is carried out.

Documentation is often considered a part of any software product. Therefore, documentation deserves the same level of testing that is applied to software. It is very essential to test documentation because good documentation not only improves ease of use, but also improves customer satisfaction. If the documentation is poor, it can lead to extra work and costs for the vendor support desk, and it might put the software producer in legal troubles. Some of the important characteristics that any documentation testing program should include are:

1. Frequent and early testing
2. Assessment of accuracy and ease of use
3. Evaluation by people who did not write the documentation

Therefore, software testing process is incomplete without performing an efficient documentation testing.

7.1.1 Types of Software Documentation

Documentation contains information on software and its components. This information makes the software user friendly and makes it easier to use the software.

Testing text based documentation is simpler compared to other forms of documentation. The tester has to make sure that the information provided is correct and is meant for that particular group of audience for which it is developed. However, when some software elements are also part of the software documentation then it becomes very important to carry out testing effectively.

The following are the elements of software that are part of software documentation testing:

1. Packaging Text and Graphics
2. Marketing Material and Other Inserts
3. Warranty/Registration
4. End User License Agreement (EULA)
5. Labels and Stickers
6. Installation and Setup Instructions
7. User's Manual

8. Online Help
9. Tutorials, Wizards, and Computer Based Training
10. Samples, Examples, and Templates
11. Error Messages

Packaging Text and Graphics

Packaging text and graphics are the text and graphics that are printed on the package of a product. The user receives a software or a hardware product inside a package. This package can be made using a box, carton, wrapped with paper or plastic, and so on. This packaging makes sure that the product inside it is well protected. The text and graphics used on the box or packaging conveys information such as company and product name, company logo, manufacturing data, and so on. The text style of each organization varies from one organization to another and also from one product to another. The kind of content and placement of content must be checked before it is printed on the package since this conveys the primary information to the users about the product.

Example: The mobile phones are usually packed and sold in a box. These boxes not only contain the mobile phone but also the accessories such as USB connector, mobile charger, ear phones, and so on. User manual and mobile suite CD are the two other important things that come with the mobile phone package. Some information related to the mobile can also be seen on the package.

Notes

While handling the package or the cartons, it is very important to provide clear instructions on how to open it, since the packaging technique used to pack the product inside the cartons is based on the design of the box. For example, where to cut the package to open the seal, whether to use scissors or blade to open the package, which is the top and bottom sides of the carton box, and so on.

For software products, the package graphics include screen shots of the software, images of special features of the software, system requirements to run the software, copyright information, and so on.

Example: Usually, on the front side of a software compact disc (CD) cover you can see the product name or code, name or logo of the organization which has developed the software, name or logo of the distributor, and so on. On the rear end of the cover, you can see major features of the software, copyright, price tag, and so on. Some mention the basic system requirements to install and run the software.

Marketing Material

Marketing collaterals help a product to sell in the market. Information that is presented on such collaterals should be informative and accurate. It serves as an important tool to convey the most important and attractive feature of the product. The information makes the user aware of the product's existence in the market and its special features. This information needs to be correct and has to be provided in an attractive manner, because it should create an interest in the customer or end user to buy the product.

Example: The pop up online advertisements that appear on the Web page comes with the list of features of the product. These advertisements are targeted to generate interest in the end users to check for other features and purchase the product and hence the content has to be accurately tested.

The sales agent uses the marketing materials to provide the customer or end user with an overview on the product. This will help in promoting the product to the users who are not aware of the product.

Example: When you visit an electronic store, the sales agent will give you a handout of a product which provides some of the key features of the product. This information helps you to know about the product quickly.

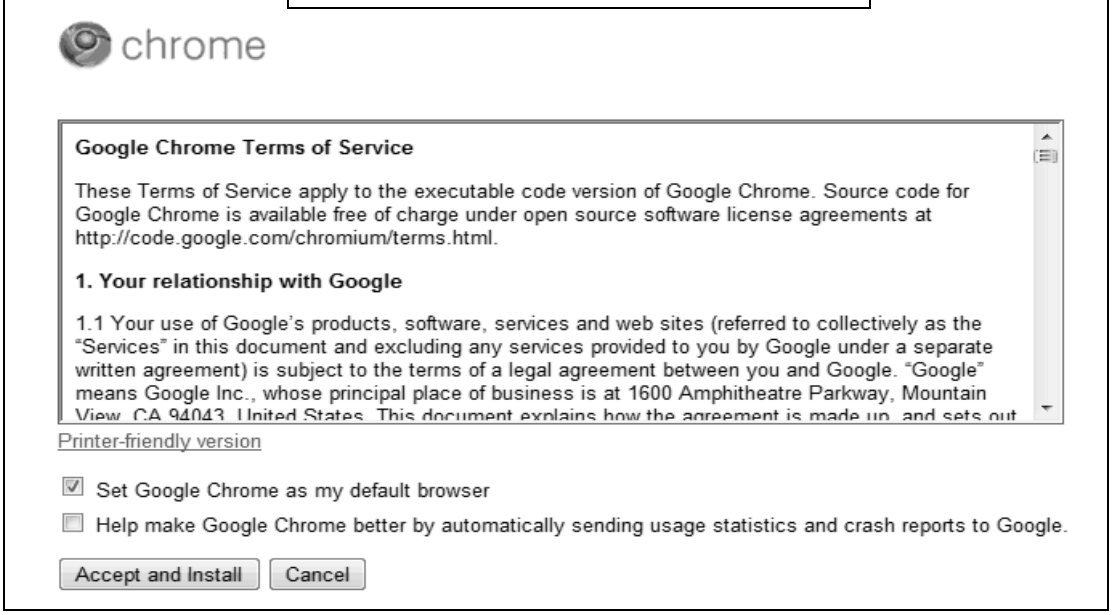
Warranty/Registration

Warranty cards are filled by users to register the software. This mainly deals with the terms and conditions for using the software. In many software products, the registration is done when the user tries to install the software. The information presented has to be easily understood by the user.

End User License Agreement (EULA)

EULA is a legal license document. The customer has to agree and follow the terms and conditions defined in the license document. The details of the license will sometimes be printed on the envelope or package of software CD or it appears as pop up while installing the software. Figure 7.1 shows an example of EULA which pops up while installing the Google's Chrome browser.

Figure 7.1: Chrome End User License Agreement



Source:http://www.google.com/chrome/eula.html?hl=en&brand=CHNG&utm_source=en-hpp&utm_medium=hpp&utm_campaign=en&installdataindex=homepagepromo

Notes

The user has to read the license terms and conditions carefully before using the software, since software licensing is a legal document that every user must know. Since the vendor or manufacturer can sue the user for any misuse of the software, the conditions of the agreement should be presented in an unambiguous way to the user.

Example: The license of Windows Operating System (OS) software clearly states that duplicate copies of its software should not be made or sold or used.

Labels and Stickers

Labels and stickers are pictorial or graphical representations of the product and company icons or logos printed on the product cover, product, manual, or on the display screen of the monitor. Warranty information is also mentioned on the product cover using labels or stickers.

Example: Figure 7.2 shows the license sticker of Windows XP software.

Figure 7.2: License Stickers of Windows XP Software

Source: http://www.technibble.com/how-to-tell-what-type-of-windows-xp-cd-or-license-key-you-have/

Installation and Setup Instructions

Installation and setup instruction information is provided to help the user to install and run the software. The instructions are usually printed on the product's package material only if the installation and setup procedure is small. In case the procedure is lengthy, then it is provided in a separate manual. Such a manual is called an installation guide.

For many software products, the procedure for installation and setup comes automatically when the user initiates the installation process.

Example: When you install the Windows operating system, the instructions for installing the software is stated clearly step by step. The user can follow the instructions appearing on the screen and install the software.

Notes

Many organizations make the installation procedure manuals available online. The user can download these manuals or read it online and carry out the process of installation.

User Manual

This is the most useful and mandatory document that all software products must accompany when the product is sold to a customer. This is available as both printed and online material. The manual provides all the information that the user would like to know about the software. This enables the user to acquire knowledge about the software functionalities and features.

Example: The Microsoft Word 2003 user manual provides users step by step instruction of how to use and work with various features and applications that is present in the software. This information helps the user to learn the software without any external assistance.

Today, many organizations provide only limited information in printed manuals. This information will help you to know and use the basic features and applications of the software. However, the more detailed manuals are made available online. The users can visit the respective organization's Web site to download or view the manuals.

Online Help

Online help is used along with the manual. It is easy to use and search for specific information easily and quickly, since it is indexed. Most online help allow users to ask the question in simple English language. The user types the question in a text box area to get a related reply. The reply could be a list of the various options or activities related to the question asked. The figure 7.3 shows the Online Help Webpage of Microsoft Word application.

Figure 7.3: Microsoft Word Online Help Webpage

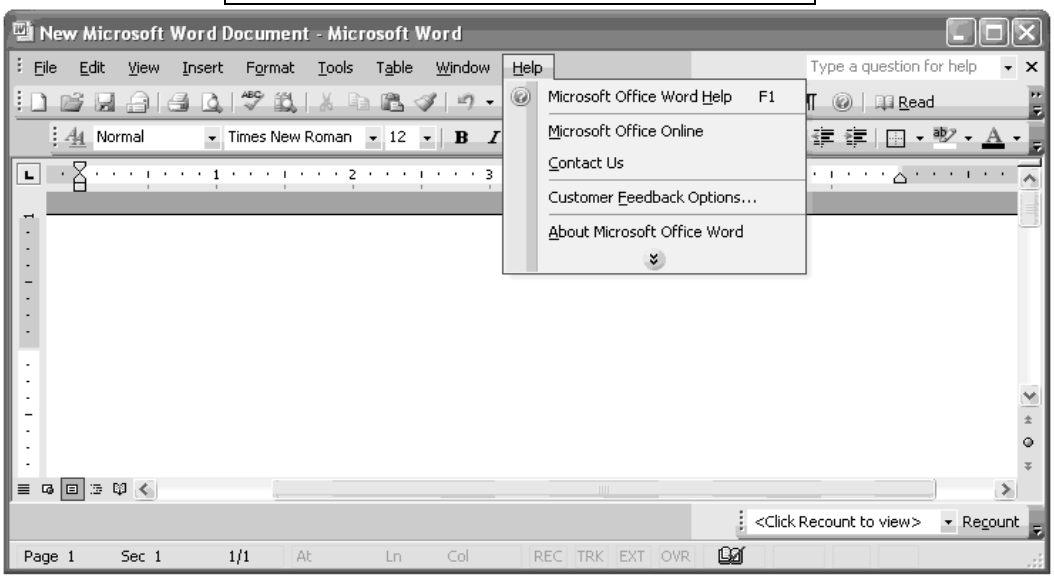
Notes

Many software products come with an inbuilt help option, which the user can use as online help. To use an online help, the user must have the Internet connection, but the inbuilt help option does not require the Internet connection as the in-built option is a default feature of the software product.

Example: The Windows online help enables you to search for information online from the working window. Windows also provides an inbuilt help option which works similar to an online help. Both these options can be accessed from the menu. Microsoft Office Word 2003 Help is the inbuilt help option and Microsoft Office Online is for online help. As you can see in figure 7.4, Microsoft's Word Window has both online help and inbuilt help in its menu -- **Help**.

Figure 7.4 shows Help options provided by Microsoft Word 2003.

Figure 7.4: Online Help and Inbuilt Help Option in Microsoft Word 2003



Tutorials, Wizards, and Computer Based Training (CBT)

This type of documentation consists of written documents and programming codes. They provide users with the useful information about the software that enables users to quickly learn steps or procedures to use the software. These documents are made available as online help, so that the users can access it easily. They also have high level macros which are programs that assist the users to perform a specific task. These macros work along with the online help system. This enables the user to quickly search and get the required information.

Example: While installing Microsoft Office Suite, the wizard that appears will help you to install the suite. It provides the information about the activity that you need to perform at every instance of the installation process. Therefore, the entire installation information is provided to you during the installation of the suite.
The Office Assistant is a Microsoft Office feature in Windows 2003, which assists users with the help of an interactive animated character that interfaces with the Office help content.

Did you know? In Mac Operating System X, wizards are called as **assistants**. Other names for wizards include the Setup Assistant which appears during the initial boot up of the Macintosh. In Microsoft Windows it is called New Connection Wizard. GNOME refers to its wizards as Druids.

Samples, Examples, and Templates

Software has certain inbuilt application samples that the user can refer or use to build his own application. Some software also provides templates which are pre-formatted examples based on which other applications can be developed.

Example: Microsoft Power point has a number of inbuilt templates for slide layout and design, which the user can apply to his presentation. These inbuilt templates come along with the software package.

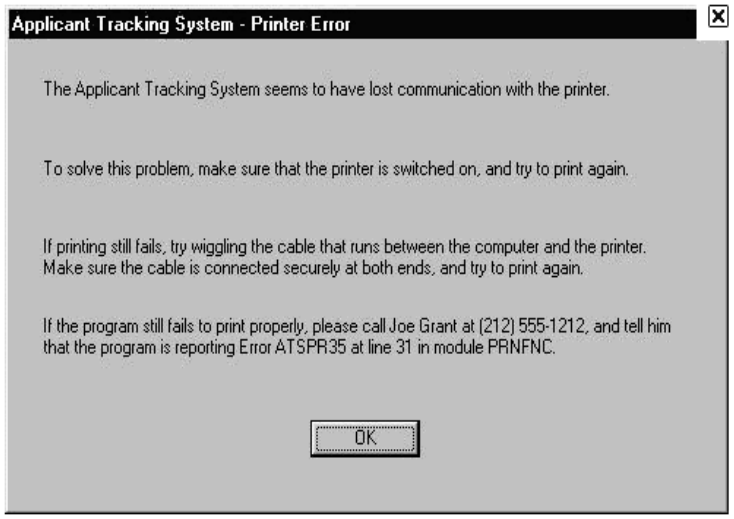
Example: Microsoft Word has a large collection of clip art images by default, which can be used in the word document. These clip art images are readily available to the users in Microsoft Office Suite.

Error Messages

Information about the error messages are important part of any software documentation. The software displays the error messages when it encounters unusual or exceptional events. These events can be triggered while using the software or hardware incorrectly or due to software related problems. The users must be able to understand the error message to know the actual problem that the system has encountered. This enables them to take necessary action to resolve the error quickly.

Example: You might have observed a Printer Error message on your Windows operating system while trying to print. This may be due to various reasons like the printer might not be connected to the CPU, the cable might not be properly plugged in on either sides or printer might not be switched on, and so on. Your operating system will display the Printer Error message as shown in figure 7.5.

Figure 7.5: Printer Error Message



7.1.2 Importance of Documentation Testing

Documentation testing is an important activity which enhances the quality of the various elements of software application. It is a well-known fact that documentation is very essential at all stages of any software development process. The documentation that is provided to the end users provides information about the product and also acts as a marketing tool for product promotion. Any error in this documentation will not only impact the product sales but also the reputation of the organization.

Example: Let us assume that a typical software product released to the market by an organization can run with a minimum of 256 Mega Bytes (MB) of Random Access Memory (RAM). However, in the installation manual of the software product, if it is mentioned that the minimum RAM required for the software to run is 512 MB, then it is a serious mistake in the documentation. This mistake is considered as an error, which the software tester must rectify and make necessary correction.

If the documentation of the software is good, then it contributes to the product in the following ways:

(a) ***It Improves the Software's Usability:*** The documentation helps the user to know how the software should be used. If it communicates the required information effectively, then the user will be able to use the software easily without much difficulty.

(b) ***It Improves the Software Reliability:*** When the user reads the documentation he/she will come to know about the various features and applications the software has. The user will read the documentation and judge the software functionalities based on the information provided. Any errors in the documentation will certainly result in poor reliability. Therefore, it is very important to test the documentation against the software to find errors in it.

(c) ***It Decreases the Product Support Cost:*** The cost incurred on the error found by the customer is 10 to 100 times more than the cost incurred to find the same error before it is released to the market. If the documentation fails to communicate the information clearly to the users, then the user will be confused and will face problems while using the software. Moreover, the organization which has developed the software has to provide customer support to resolve the problems that the customer is facing – which proves to be expensive. Therefore, in order to overcome such problems, good documentation must be provided with the software product. This documentation helps the customers to easily understand and use even the most difficult application or feature of the software.

Documentation testing checks for the correctness of facts and figures mentioned in the documentation. Testing must make sure that all the instructional steps are explained clearly and effectively. The testing must also check whether or not the documentation meets all the requirements of the end user. Thus, documentation testing helps in finding errors in the documentation and helps in providing correct, accurate, and effective information to the users. It improves customer satisfaction

7.2 Security Testing

Security testing must not be confused with safety testing. The aspect of security deals with how well the software is protected from external elements such as hackers who make use of virus to affect the normal operation of the software.

Example: A type of computer viruses named resident virus such as Randex, CMJ, Meve, and MrKlunky attack the RAM memory of the system. From RAM, it affects the normal operation of the entire operating system. It will corrupt the files currently used by the operating system.

A tester who is performing a software security test must apply risk based approach to find the bugs in the software architecture and design. The testers must have the mindset of an attacker (hacker) to find bugs related to software security. This requires identifying the risks that the software is prone to during an attack and creating test cases based on the risks identified. Such test cases will enable the tester to focus on a particular area of code where the possible attack can be successful.

Example: Banks use smart card technology to overcome fraud and misuse of account holder's freedom. These smart cards use the Crypto System to carryout transactions and verify the identities of the cardholder and the bank. The card holder information and account have to be very well protected, as the transactions happen online and there is always a possible risk of the system being hacked or a virus attack. If any such incident happens, the bank will lose valuable information and also its account holder's money. Testing for possible risk in the software will help to overcome and stop any attack on the software and prevent theft of information.

Software security testing tests the software behavior when the software is attacked by some external element. Sometimes, software failure occurs without any external interference. This may occur due to weak design or coding. Therefore, software security aims to protect the software from any such failures. Any software designed with poor logic is prone to external attack from hackers.

Hackers make use of weak codes in the software to carry out an attack on the software and cause software failure. Therefore, any such vulnerability is considered as a bug. It is the responsibility of the tester to make sure that any such bug in the software is detected and reported to the development team.

Information and services are the two important aspects of software security. Protecting both the information and services from possible external attack is vital. Therefore, risk analysis should be carried out at the design level so as to identify any potential security problems and resolve them at the earliest.

Example: In 2008, hackers from Russia robbed an Automatic Teller Machine (ATM) in New York. The hackers were able to obtain the PIN numbers of customers after they hacked the main server of the ATM company. They managed to steal $180,000 from ATM machine.

Notes

Software security testers perform many different tasks to manage risks related to software security such as:

1. Creating security abuse/misuse cases.
2. Listing normative security requirements.
3. Performing architectural risk analysis.
4. Building risk-based security test plans.
5. Wielding static analysis tools.
6. Performing security tests.
7. Performing penetration testing in the final environment.
8. Cleaning up after security breaches.

Basically, security testing involves two approaches:

1. Testing software security mechanisms to check whether the functionalities of this mechanism are properly implemented during the software product design and coding.
2. Performing the risk based security testing with the perspective of an attacker and developing test cases to check for possible risks.

The tester should make sure that they identify all possible bugs in the software to minimize the risk of software being prone to any external attack from hackers.

7.2.1 Threat Modeling

The process of assessing and documenting the system's vulnerability to security risks is known as security threat modeling. It enables the organization or developers to understand the various threats that the system can face. To achieve this, the tester will analyze the system with an attacker's perspective. This enables the tester to identify the possible threats and rate them based on order of threat severity i.e., greater the risk, higher the order of severity. This enables the organization or developers to address the risk which can pose a greater risk to the system.

Threat modeling is a highly structured and organized approach of threat identification. It is also a highly cost effective approach, which helps to identify the possible threats efficiently and effectively. The principle behind the model is that "one cannot make the system secure until they know the threats the system can face".

Threat modeling helps to identify and tackle the risk during software evolution. It should be carried out at every level of software development life cycle, as identifying and resolving threats of a fully developed product require both time and cost.

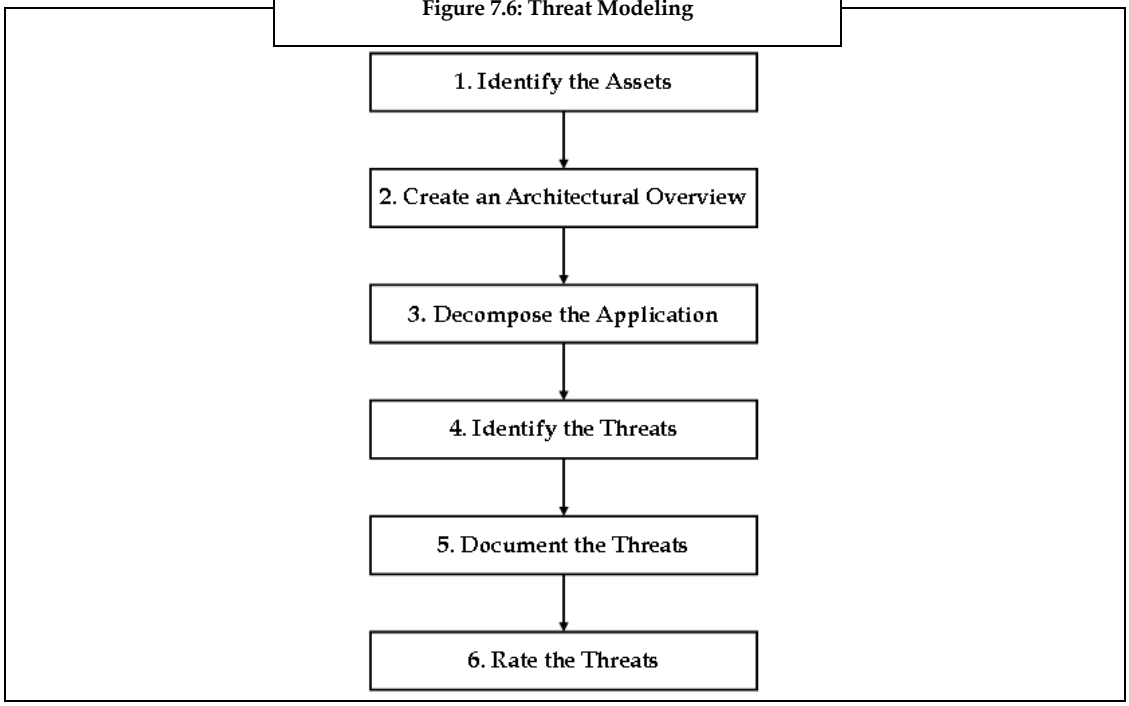
Notes

Business requirements keep changing during the product development life cycle. The developers have to make modifications to the actual specifications and design during the product development. Carrying out the process of threat modeling regularly will help in identifying the risks more efficiently and effectively and resolve them as soon as possible.

As shown in figure 7.6, the process of threat modeling is divided into six stages.

1. Identify Assets
2. Create an Architecture Overview
3. Decompose the Application
4. Identify the Threats
5. Document the Threats
6. Rate the Threats

Figure 7.6: Threat Modeling



Identify Assets

At this stage, the tester will identify all the assets that are associated with the system that has to be protected.

Example: Confidential customer data, orders, and employee database are some areas that are identified.

Create an Architecture Overview

The tester will use representations like simple diagrams, tables, and graphical representation to describe and document the architecture of the system. This helps the tester to understand the actual working of the system.

The following tasks are performed during this step.

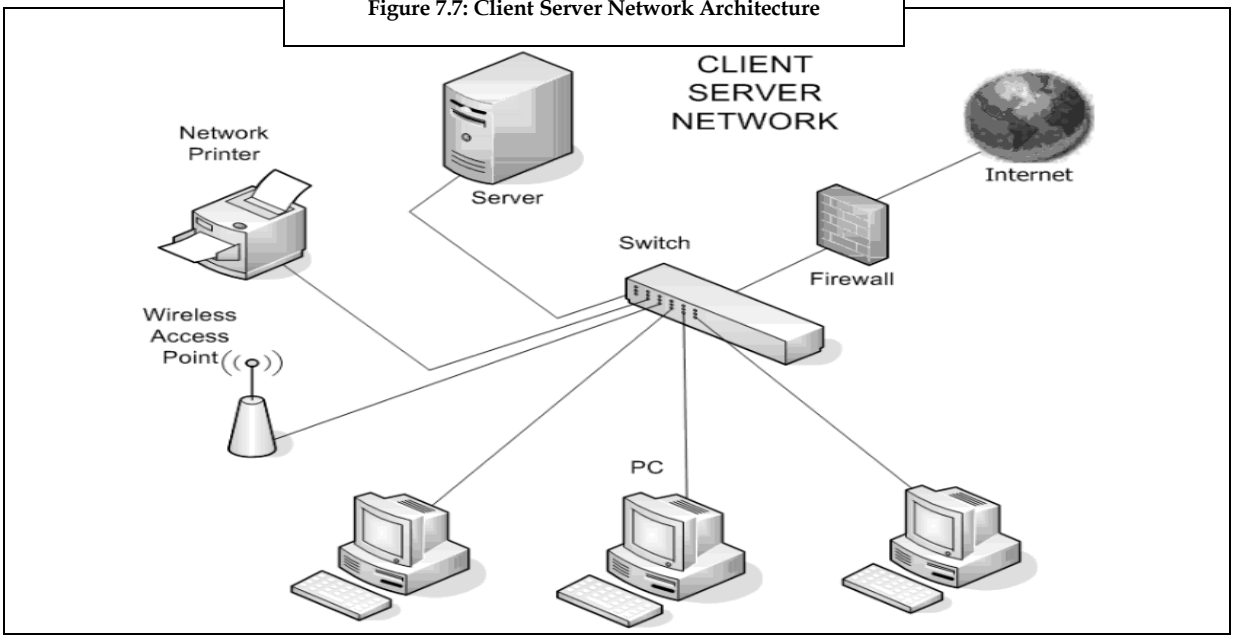
(a) Identify what the system does and how it accesses the various subsystems.

Example: In an employee database management system, the employee views various subsystems such as financial data, personal data, and project data. The managers view employee details, employee log in times, and so on.

(b) Create an architecture diagram of the system. The architecture diagram includes subsystems, boundaries of operation, data flow channels, and so on.

Example: The network architecture overview comprises various computers that are connected to the local server and access the data from the protected system. Figure 7.7 illustrates an architecture overview of a client server network system. It shows the channels used to communicate between various systems and subsystems.

Figure 7.7: Client Server Network Architecture



Source: http://www.directassist.net/service-plan.htm

(c) Identify the technologies associated with the system as these are the technologies that are used to implement the system.

Decompose the Application

Decomposing the application refers to understanding the platform on which the system operates and designing appropriate standards.

Example: In a network system, the tester identifies the network and host infrastructure design.

This helps them to create a security profile for the application. This profile is used to detect the vulnerabilities in the various area of the system such as design, implementation, and configuration.

Identify the Threats

The tester should think and act like a hacker and find the vulnerabilities in the system. The tester should have the knowledge of the entire system architecture and potential vulnerabilities of the system. This enables him/her to identify the threats that could affect the system.

Example: In a network system the tester identifies the threats such as network threats, host threats, application threats, and so on.

Document the Threats

The tester will use a common document or template to record all the threats that he/she has detected in the system. A typical template consists of:

1. ***Threat Description:*** It defines the threat that has been detected.
2. ***Threat Target:*** It specifies the actual target of the attacker.
3. ***Risk:*** This is used to mention the priority based on the criticality of the risk.
4. ***Attack Techniques:*** These are the techniques that are used by the attackers to carry out the attack.
5. ***Attack Techniques:*** These are the techniques that are used by the attackers to carry out the attack.

Example: The tester has detected two threats in the system which he/she has to document. The figure 7.8 shows the template that the tester uses to record the threat he/she has detected. The template has five rows which are threat description, threat target, risk, attack techniques, and counter measures.

Figure 7.8: Template to Record Threat

Threat Description	Attacker obtains authentication credentials by monitoring the network
Threat Target	Web application user authentication process
Risk	
Attack Techniques	Use of network monitoring software
Countermeasures	Use SSL to provide encrypted channel

Table 3.6 Threat 2

Threat Description	Injection of SQL commands
Threat Target	Data access component
Risk	
Attack Techniques	Attacker appends AQL commands to user name, which is used to form a SQL query
Countermeasures	Use a regular expression to validate the user name, and use a stored procedure that uses parameters to access the database.

Source: http://msdn.microsoft.com/en-us/library/aa302419.aspx

Notes At this level the risk column is left empty since the risk ratings are given in next level based on the criticality of the risk.

Rate the Threats

This is the last stage of the process. The threats are rated using simple high, medium, or low ratings. This rating is based on the risk value which is calculated using the following simple formula:

Risk= Probability * Damage Potential

where,

(a) Probability represents the probability of occurrence of the threat. This is rated using the scale of 1 to 10, where 1 indicates a threat that is very rare to occur and 10 indicates a common and definite threat.

(b) Damage Potential represents the damage caused by the threat. This is also rated using the scale of 1 to 10 where 1 represents minimal damage and 10 represents a catastrophe to the system.

Example: If Probability = 5 and Damage Potential = 3 then Risk = 15.
Similarly, if Probability = 7 and Damage Potential = 3 then Risk = 21.

Based on the risk value calculation, the threat rating high, medium, or low is given to the threat.

Finally, a detailed documentation of the security aspects of the system that lists and rates all possible threats is prepared. This helps the development team to focus and resolve the detected threats.

Notes

The process of threat modeling is not only carried out by a tester but it is done by the project manager because this model demands the involvement of all the members associated with the software development such as project manager, developer, tester, technical writer, and so on.

7.2.2 Buffer Overrun

Buffer overrun is one of the most common security problems today. Buffer is a memory area shared by microprocessors and other hardware devices. Buffer overrun is caused by buffer overflow. Buffer overflow occurs when the value saved in the buffer exceeds the size of the buffer memory. The excess memory values either overwrite the existing buffer memory or other buffer memory. Such buffer overruns might crash the system.

Example: Software developed using programming languages such as C and C++ have no built-in protection against accessing or overwriting data during buffer overruns. This is because there are no inbuilt functions to check whether or not the data written to an array is within the defined boundary limits of the array.

Bugs are caused when excess data is saved in a limited buffer memory. This will overwrite the adjacent stack or memory of the buffer and will often cause the program to behave incorrectly or crash. This not only affects the execution of the program, but also throws serious security vulnerability.

Example: A hacker can overrun the buffer of the running program by supplying un-trusted data and can corrupt the stack. Later the hacker could overwrite the buffer with an un-trusted executable code and thereby can take control of the system.

Did you know? In 1997, the Carnegie Mellon University issued 28 advisories related to security vulnerabilities. Out of these 28, 15 were related to buffer overrun vulnerabilities identified in commonly used programs and facilities in UNIX systems.

7.2.3 Safe String Functions

Safe string functions are functions that are used to overcome buffer overruns in software programs. Poor handling of buffers causes a buffer overrun and leads to many system security problems. Such poor handling is related to string manipulation operations. The safe string functions perform extra

processing of the input data for proper handling of buffers in the software. Since, C or C++ language do not have proper control over the data being stored in the buffer, these strings replace the standard string functions that are available such as strcat, strcpy, sprintf, and so on.

Example: The **strcpy** is a function used to copy the string value from one variable to another variable. If the destination variable array length is small compared to the source variable, then the problem of overflow occurs in the destination variable.

The new set of string functions, i.e., the safe string functions that help to overcome the overrun problem are developed by Microsoft for the Windows XP SP1, Windows Driver Device Kit and platform Software Development Kit.

Some of the advantages of using safe string functions in a program are:

1. Along with the input data, the functions also receive the destination buffer's size as input. This makes sure that the destination buffer does not overrun if the input data exceeds the normal size of the destination buffer.
2. The string functions terminate all output strings with a Null character, which indicates the end of the string. Other functions using these strings can assume that they will encounter null character. Therefore, the data before the null character is a valid data and null character terminates the string without allowing it to run indefinitely.
3. NTSTATUS value is returned by all safe string functions. This value indicates the calling function that the safe string function has performed the operation successfully.
4. The safe string functions are available in two versions. One version supports double-byte Unicode characters and the other supports single-byte American Standard Code for Information Interchange characters.

When the tester performs the white box test of the software, then the tester has to check for unsafe strings in the program code and how they are used in the program logic. This enables to develop test cases to check whether or not these unsafe string functions cause overruns. It is advisable that the programmers extensively use safe string functions instead of unsafe functions.

Table 7.1 shows the list of various safe and unsafe string functions developed by Microsoft.

Table 7.1: Safe String Functions List

Purpose	Unsafe Strings Functions	Safe String Functions
Concatenate two strings.	strcat wcscat	RtlStringCbCat RtlStringCbCatEx RtlStringCchCat RtlStringCchCatEx
Concatenate two byte-counted strings, while limiting the size of the appended string.	strncat wcsncat	RtlStringCbCatN RtlStringCbCatNEx RtlStringCchCatN RtlStringCchCatNEx
Concatenate two byte-counted strings, while limiting the size of the appended string.	strcpy wcscpy	RtlStringCbCopy RtlStringCbCopyEx RtlStringCchCopy RtlStringCchCopyEx
Copy a byte-counted string into a buffer, while limiting the size of the copied string.	strncpy wcsncpy	RtlStringCbCopyN RtlStringCbCopyNEx RtlStringCchCopyN RtlStringCchCopyNEx

Contd…

Determine the length of a supplied string.	strlen wcslen	RtlStringCbLength RtlStringCchLength
Create a formatted text string that is based on a format string and a set of additional function arguments.	sprintf swprintf _snprintf _snwprintf	RtlStringCbPrintf RtlStringCbPrintfEx RtlStringCchPrintf RtlStringCchPrintfEx
Create a formatted text string that is based on a format string and one additional function argument.	vsprintf vswprintf _vsnprintf _vsnwprintf	RtlStringCbVPrintf RtlStringCbVPrintfEx RtlStringCchVPrintf RtlStringCchVPrintfEx

Source: http://www.osronline.com/ddkx/kmarch/other_9bqf.htm


Notes

It is the responsibility of the programmer to use the inbuilt string function of C or C++ languages properly to overcome any overruns problem. Proper logic and correct usage of strings can overcome the overrun problem.

7.2.4 Computer Forensics

Computer forensics is also called as cyber forensics. This is a technique of computer investigation and analysis that is used to gather substantial evidence against a cyber crime for presenting it in a court of law. The main aim of computer forensics is to conduct a structured investigation of a cyber crime to find out what happened and who was responsible for it. This is done to protect the security of software.

Computer forensics deals with identifying and solving crimes that are carried out by using computer technology. The governments across the globe have imposed many laws to check cyber crimes. However, lack of evidence has made it difficult to prosecute the people responsible for the crimes. Computer forensics helps to overcome such difficulties. It helps to gather evidence to take legal actions against those who carry out such crimes.


Did you know?

International Data Corporation (IDC), in the year 2005, reported that "the market for intrusion-detection and vulnerability-assessment software will reach 1.45 billion dollars in 2006" (US-CERT, 2005).


Notes

Some of the major reasons for criminal activities in computer are:

Unauthorized use of username and password.

1. Accessing other users' computer via the internet.
2. Releasing virus to other computers.
3. Harassment and stalking in cyberspace.
4. E-mail Fraud.
5. Theft of company documents.

The tester should check for security vulnerability issues related to test software from a computer forensic perspective. Sometimes, hackers do not really need to break into your system to steal the data, since some data can be easily accessed. If the hacker knows where exactly to look for a particular data then he/she can easily get the data from the software.


Example:

When you download any file or picture from the internet-- by default, all the files and pictures are saved in "Temporary Internet Files" folder on Windows operating system. In case you have accessed any confidential information on the Internet, the same would also be saved in the "Temporary Internet Files", and a hacker can easily view and use this file.

Any unprotected data that is available for other users is called latent data. It is the responsibility of the tester to assess whether or not any such latent data can cause security vulnerability. If yes, then necessary measures have to be taken to prevent it from occurring.


Notes

A tester who is very well aware of the security vulnerabilities of the system can provide vital information to computer forensics to know exactly how the software security could be breached. This helps to find out the possible ways an attacker could have carried out the attack.


Case Study

National Widgets Website Security Problem

National widgets wanted to build a Web site for its users, and it approached Front End Associates to develop a Web site for them. Front End Associates developed a highly impressive Web site for National widgets.

National widgets deployed the Web site developed by Front End Associates and for about 18 months the Web site operated without any problem. However, some of the employees of National Widgets raised security concerns in the Web site. National Widgets brought the issue to Front End Associates' notice and asked them to fix the problem for free. However, the Front End Associates were not ready to fix the problem for free and did not respond to National widgets' request properly.

Later, Front End replied to National Widgets saying that there were no security problems in the software that they had developed. It also justified its stand by saying that they had hired Web site security testing experts to carry out security tests on the software and provided a detailed report. National widgets decided to verify the test reports that the team of testing experts had prepared after testing the Web site. National widgets had to take the support of its lawyers to view the test report which was with the Front End. After analyzing the report it was noticed that they had not conducted an effective testing of the software. The company that Front End hired to perform security testing had simply run a few scanning tools to check for minor issues in the software. They did not perform an effective testing to find security vulnerabilities in the software.

National widgets planned to conduct an independent testing of the software and assembled a group of experts to carry out the test. For this, National widgets asked Front End to provide the source code of the Web site that Front End had developed for them. However, Front End refused to give the source code to National Widgets, saying it will cause copyright issue. National widgets, with the help of law, was able to decompile the code and perform a code review. After a thorough testing of the software, more than six serious problems related to security were found. These problems were due to poor design that Front End had adopted during the initial stages of Web page development. The problems were so severe that it required both time and cost to make necessary changes. Therefore, National widgets raised a request to Front End to fix the problem without any additional cost. However, Front End partially acknowledged the defects in the Web page, but was not ready to accept the mistake completely. This became a legal issue and both the companies went to court to solve the dispute.

This problem had a direct impact on both the companies. The companies lost lakhs of rupees by paying legal fee, productivity was hindered, and reputation was damaged. Along with this, both the companies had to spend huge time and money for answering each other's queries, producing documents, re-testing of the software, and trial. Even though Front End was the most affected in terms of loss of revenue and reputation, National widgets also saw some setbacks due to this issue.

Questions

1. What was the problem that National faced and what was the reason behind it?
2. Do you think Front End was responsible for developing such a Website? Justify.

Adapted from
http://www.owasp.org/index.php/Secure_software_contracting_hypothetical_case_study#Conclusions

7.3 Summary

- Product documentation provides users the information about the product specifications. This helps the users to know about the product and its features. Thus, it enables the customers to use the product easily.
- The tester performs documentation testing to check for any errors in the document. Since documentation errors will not only convey incorrect or wrong information to the users it will also bring down the reputation of the company.
- The various software components of documentation are Packaging Text and Graphics, Marketing Material, Ads and Other Inserts, Warranty/Registration, End User License Agreement, Labels and Stickers, Installation and Setup instructions, User's Manual, Online Help, Tutorials, Wizards, and Computer Based Training (CBT), Samples, Examples, and Templates, and Error Messages.
- Documentation testing helps to improve the usability and reliability of a software product. It also helps the organization to reduce the product support cost.
- Security testing is the most important aspect of software testing. This enables the tester to find the system's vulnerability to security risks.
- Security threat modeling helps to analyze the system in a structured way, so as to find the threats that the system faces with respect to security. This model not only detects the threats, but it also documents the threats found and rates them based on the severity of the threat.
- Buffer overrun is one of the most popular bugs that the hackers use to attack the system. It is a major security threat for any software product.
- The usages of safe string function have enabled the developers to overcome the problem of buffer overrun. The tester has to make sure that the developers use these functions to develop their programs.
- Testers must test the software for any latent data available in it, since this data can cause issues related to software security.

7.4 Keywords

Crypto System: Any computer system that involves cryptography is called as crypto system. Cryptography is an art of studying hidden, coded, or encrypted information.

Unicode: Binary codes that are used to represent text or script characters in computer programming languages.

Virus: A computer program that can copy itself and infect a computer.

Vulnerability: Susceptibility to attack.

Warranty: A written assurance that some product or service will be provided or will meet certain specifications.

7.5 Self Assessment

1. State whether the following statements are true or false.
   (a) Documentation meets its objective only if it provides necessary and complete information to the end users or customers.
   (b) The details of the license will sometimes be printed on the envelope or package of software CD.
   (c) Today, many organizations provide the entire information about a product using printed manuals.
   (d) Threat modeling is a highly structured and organized approach of threat correction.

(e) The tester has the knowledge of the entire system architecture and potential vulnerabilities of the system.

(f) The main aim of computer forensics is to conduct a structured investigation of a cyber crime to find out what happened and who was responsible for it.

2. Fill in the blanks

(a) _____ material creates interest in the customer or end user to buy the product.

(b) In many software products, _____ is done when the user tries to install the software.

(c) The software displays the _____ when it encounters unusual or exceptional events.

(d) _____ make use of weak codes in the software to carry out an attack on the software.

(e) The tester will use a common _____ to record all the threats that he/she has detected in the system.

(f) The _____ perform extra processing of the input data for proper handling of buffers in the software.

3. Select a suitable choice for every question:

(a) Identify which among the following is not documentation.

(i) Labels and stickers (ii) Tutorials and wizards

(iii) End User License Agreement (iv) User feedback report

(b) What is a legal document?

(i) Warranty (ii) End User License Agreement

(iii) Registration form (iv) Error messages

(c) What is called as short version of a user manual?

(i) Tutorials (ii) Wizards (iii) Online help (iv) Installation guide

(d) What is the most important aspect of software security?

(i) Cost (ii) Time (iii) Information (iv) Quality

(e) Which is the step that follows soon after identifying the threats in software threat modeling?

(i) Identify assets (ii) Decompose the application

(iii) Rate the threats (iv) Document the threats

7.6 Review Questions

1. Do you believe that documentation is a window that provides user a complete view of the product?
2. Documentation testing is a crucial element of any software testing process. Justify
3. Do you think software components can be called as documentation? If yes, explain with examples.
4. "Software security testing tests the software behavior when the software is attacked by some external element." What do you consider as external element and how would you ensure testing the same?

5. "Buffer overrun is one of the most common security problems today." What kind of problems do you oversee with overrun and how can they be overcome?
6. "Threat modeling should be carried out at every level of software development life cycle." How is this done?
7. "Good documentation contributes to the productivity of the organization." Explain.
8. Is there a need for software security testing? Justify.
9. "While rating the threats, a small calculation has to be performed to find the risk value." Explain with an example how you will carry out the calculation.
10. If you are a software tester, what are the approaches that you will follow when it comes to security testing?
11. "Security threat modeling is a structured process that involves various steps to carry out the process of threat detection." Explain.

Answers: Self Assessment

1. (a) True (b) True (c) False (d) False (e) True (f) True
2. (a) Marketing (b) Registration (c) Error messages
   (d) Hackers (e) Template (f) Safe string functions
3. (a) User feedback report (b) End Users License Agreement
   (c) Online help (d) Information (e) Document the threats

7.7 Further Readings


Books

Patton R, Software Testing-Second Edition, SAMS Publishing, USA
Hutcheson, & Marnie L. (2003). Software Testing Fundamentals, USA: Wiley Publishing Inc.


Online link

http://www.ciol.com/Testing/Feature/Know-more-about-documentation-testing/30608107510/0/
http://www.articlesbase.com/business-opportunities-articles/importance-of-documentation-in-software-testing-3801952.html
http://msdn.microsoft.com/en-us/library/aa302419.aspx
http://www.osronline.com/ddkx/kmarch/other_9bqf.htm
http://msdn.microsoft.com/en-us/library/ff565508.aspx
http://www.computerforensics1.com/
http://www.agilemodeling.com/artifacts/securityThreatModel.htm

Unit 8: Web Site Testing

CONTENTS

Objectives

After studying this unit, you will be able to:

- Discuss the fundamentals of Web page testing
- Explain black box testing with respect to Web page
- Describe white box testing and distinguish gray box testing
- Outline configuration and compatibility testing for Web pages

Introduction

Web site testing is as important as any software or application testing. Web site testing refers to software testing that focuses mainly on Web applications.

A Web site is a collection of a number of pages, which includes texts, graphic images, links, sounds, and other elements. A Web site can be defined as a collection of one or more Web pages grouped under the same domain. It must contain a domain name and a Web host.


Notes

The individual pages of a Web site are called Web pages. A Web page can be created using Hyper Text Mark-up Language (HTML).

The domain name is the address of a Web site.


Example: www.triumphindia.com is the domain name of the Web site of a company called Triumph India Software Services Private Limited.

A Web host is used to store the Web site. One can open a Web site stored in a Web host by entering the domain name of the Web site in the address bar.

The Importance of Web site Testing

Web sites are now considered a natural medium for transactions by individuals and businesses. Business sectors use Web sites to exchange views, to buy and sell goods, provide services, and so on. Web sites also provide access to information, thereby assisting the education sectors. This makes web testing with all software platforms important. It should also be ensured that the Web site is user friendly for handicapped and blind users.

Example: Importance of Web site Testing

Consider that you have designed a Web site for your company, which is compatible with or can be accessed using the browsers Windows XP and Internet Explorer. How would the customer access the Web site while using other browsers like Mozilla, Linux, or chrome? Thus, Web site testing is essential to ensure that your Web site is operational on every browser and across all platforms.

To further understand Web site testing, it is important to know the fundamentals of a Web page. In this chapter, we will discuss the fundamentals of a Web page, which include the various elements that build a Web page, which in turn helps build a Web site. We will also discuss how each element can be tested, how black box, white box, and gray box testing strategies can be applied to Web site testing, and how configuration and compatibility testing can be performed.

8.1 Web Page Fundamentals

As discussed earlier, the Web pages contain texts, pictures, sounds, videos, and hyperlinks (links which allow access to other Web pages). The data stored solely on a Compact Disk-Read Only Memory (CD-ROM) is restricted to a single computer, but Web pages are not associated with a single computer. Internet facilitates users across the world to search for information on any Web site on any computer.

The most important goal of Web pages is to present information in the most dynamic and up-to-date manner possible. Web sites have three fundamental components. They are:

1. ***Home Page:*** Home page is the default page in the Web site.
2. ***Links:*** Links are provided to link all local and remotely stored Web pages to the home page.
3. ***Content:*** Texts, graphic images, and sounds form the content.

Home Page

The first page that appears when we log in to a Web address is known as the home page. It captures the attention of the visitors and sets the tone for organization of content in the site. Usually, home page includes a header on the top representing the source name of the site. Some simple headers include only text, while others include designs along with graphic images.

Links

Links on a Web site are used to access local and remotely stored Web pages linked to the Web site. Links are used to:

1. Navigate users to other Web pages of the same site.
2. Direct the user to a different location on the same page.
3. Download files from the Web site.
4. Allow users to access other Internet tools, such as default e-mail clients like the Microsoft Outlook.

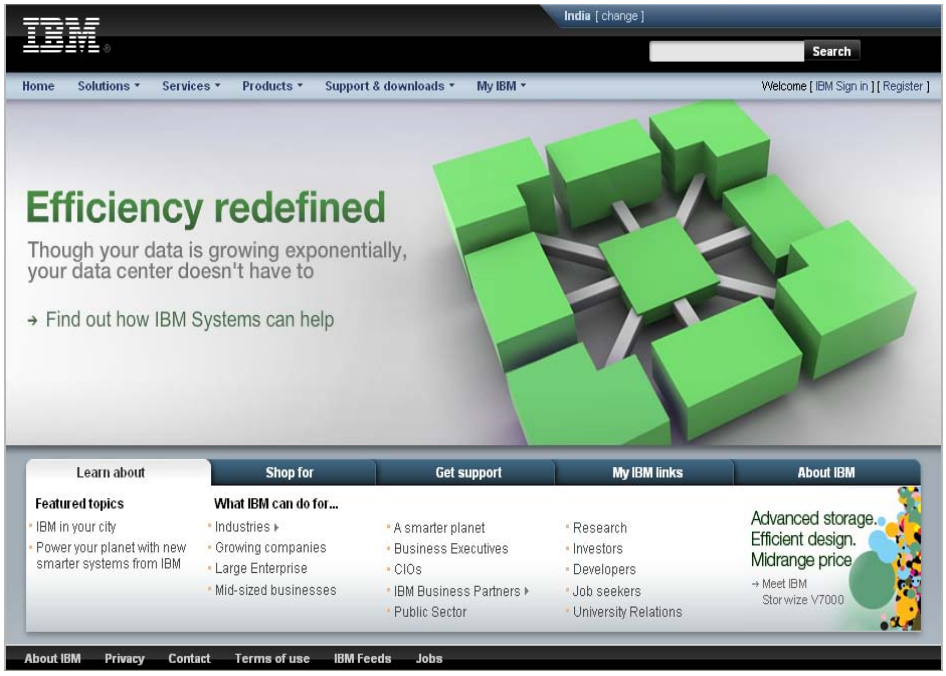
Content

Content is the most important part of a Web site. The content of a Web site can take many forms, including:

1. Text documents
2. Graphics
3. Sounds
4. Movie clips (that can be downloaded)
5. Fields (that enable the users to enter the data)
6. Advertisements (that keep rotating)
7. Text (that change dynamically)

Figure 8.1 depicts IBM's Web site that demonstrates general elements of any Web site.

Figure 8.1: Example of a Web Site



Source: (http://www.ibm.com/in/en/)

Let us now discuss the need to build a Web site and the fundamental goals of a Web site. The goal of a Web site is to provide up-to-date and accurate information to the users, partners, and employees of an organization. Effectively managed Web sites behave as an excellent platform to provide information to a wide range of audience. They also have the ability to rapidly update and modify information as needed. A Web site of any company or business helps the company or business to get promoted frequently and get new opportunities. The company or business can keep on updating its Website as and when there are any changes in its offerings, prices, business strategies etc., so as to keep its stakeholders informed.

A Web site developed to improve the business should be constructed with the following goals:

1. The matrices for success have to be defined – it may simply be the number of visits to the site or the information the business intends to share with the visitors.
2. The customers should be able to navigate through the required information with minimal clicks. They should also be able to place an order comfortably.
3. The business organization has to record the number of clicks required for basic navigation. This includes the average time spent by a potential customer who visits the site. It would be good to find out whether the potential customers are able to find the required information.

Once you build a Web site, it is necessary to test it for its requirement specifications. Let us discuss the points to be considered while testing a Web site. Web sites involve client/server applications with Web servers and browsers. While testing a Web site, importance should be given to:

1. Interactions between the pages created using scripting languages
2. Transmission Control Protocol (TCP)/Internet Protocol (IP) communications
3. Internet connections
4. Firewalls
5. Applications such as applets, java scripts, plug-in applications that run in Web pages
6. Applications such as JavaScript, database interfaces, logging applications, dynamic page generators, and so on that run on the server side

In addition to the above listed points, Web testing should also consider the significant differences between various versions of servers, browsers, platforms, connection speeds, rapidly changing technologies, and multiple standards and protocols.


Notes

Terms and Definition

1. ***Applet:*** It is a small program that does not run on its own -- instead it needs to be embedded into another application.
2. ***Java script:*** It is a type of programming language and Web pages communicate interactively through the Java scripts.
3. ***Plug-in Application:*** It is also known as a helper application, which helps a parent application by providing more instructions.
4. ***Transmission Control Protocol:*** It is a protocol developed for the internet to transmit data from one network device to another. It ensures reliable, connection based communication.
5. ***Firewall:*** It acts as a barrier between computers on a network to protect them from intruders, especially programs that destroy, tamper with, or gain access to files.

Various Elements of a Web site and Their Testing Methodologies

A Web site tester needs to test a Web site in all possible ways to make sure that the Web site provides the defined benefits. The elements to be tested and their testing methodologies are as follows:

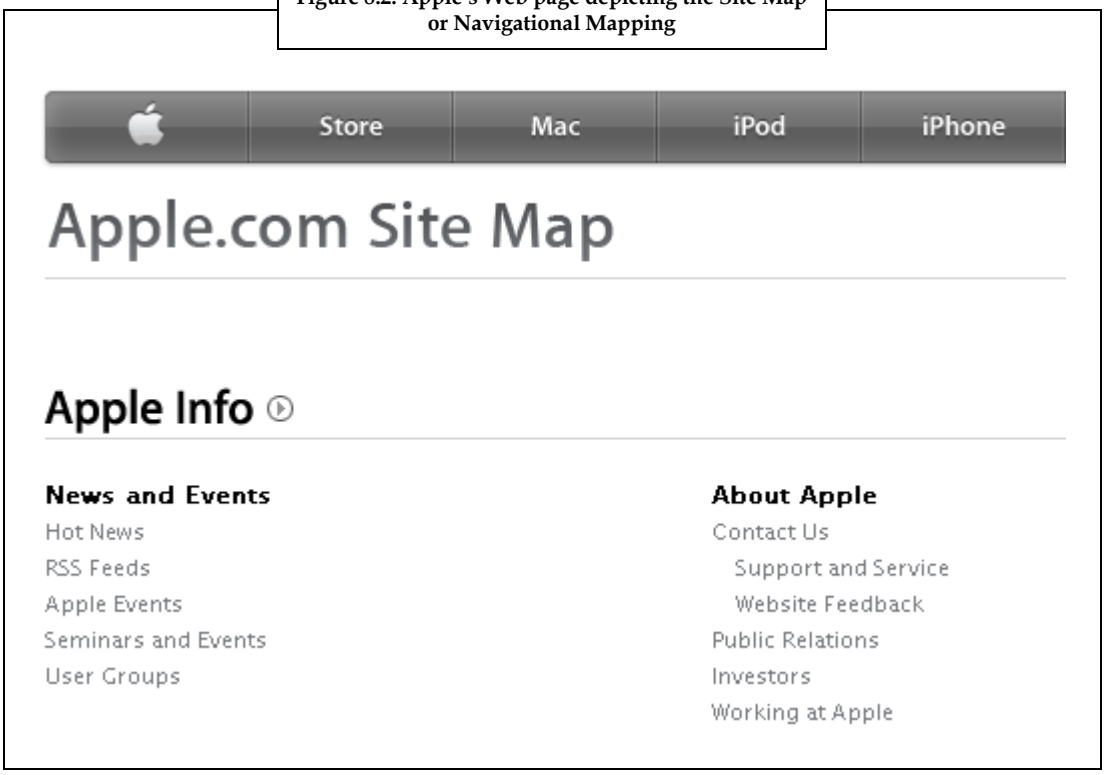
1. ***User Interface:*** When focusing on the user interface testing of your Web site, verify that the Web site is simple to use. Many believe that this is the least important area to be tested. A Web site should be tested against user interface to achieve customer satisfaction.
2. ***Instructions:*** Make sure that your Web site contains all relevant instructions. Since many people use Web sites, some may require some clarification. Even though your Web site is simple and easy to use, it is essential to provide instructions in your Web site. A tester can test those instructions against the specification document.

3. ***Site Map or Navigational Mapping:*** You need to test whether the site has a map, because the experienced users generally know where exactly they want to go, and avoid lengthy instructions.

   Hence, a site map should provide instructions to the new users to search for any information in that particular Web site.

   Example: Figure 8.2 shows Apple's Web page that contains a site map for all the pages. If a visitor wants to learn about the company, the user can go to the column 'About Apple' and choose different areas under that column.

Figure 8.2: Apple's Web page depicting the Site Map or Navigational Mapping



Source: (http://www.apple.com/sitemap/)

While testing site maps, you need to test for the following points:

(a) Is the site map appropriate?

(b) Does each link on the map really exist?

(c) Are there any links on the site that are left out on the map?

(d) Is the navigational bar present on every page of the site?

(e) Is the navigational bar consistent throughout the site?

(f) Does each link work on each page? Is it organized in an intuitive manner?

4. ***Content:*** Verify the Web site content with the public relations department to ensure that it is not subject to plagiarism and that the Web site does not have any copyright issues. Plagiarism can lead to legal issues. You also need to ensure that the site is designed in a professional way by ensuring minimal usage of bold text, big fonts, and blinking texts, as they may not appeal to the customer who might thus not revisit the Web site. Also, all the Web references should be hyper linked.

5. ***Colors/Backgrounds:*** When the sites have colors/backgrounds for pictures or texts, ensure that the user does not find it difficult to read the content. It would therefore be better to use little or no background. However, when a background is required, avoid using bright colors because patterns and pictures divert the user's attention from the content on the Web site.

Example: Consider the following two figures (figure 8.3 and 8.4) to understand why a Web site should use single or no background colors. Figure 8.3 uses no background. Hence, a user can read it easily. However, figure 8.4 uses background color, and the user may find it difficult to read the content.

Figure 8.3: Web site with no Background Color

Figure 8.4: Web site with Background Color

6. ***Images:*** A picture is worth a thousand words, so use it to convey messages to the users. However, though an image may be convincing, it may choke the bandwidth due to its size. Hence, it is an important concern for designers and testers to check whether the images add value to each page or

simply consume bandwidth. Use different file types for images such as jpeg, png, bitmap, and so on.

7. ***Tables:*** Verify the position of tables for convenience of use, because sometimes the users may need to constantly keep scrolling to access the information.

Example: If a marketing company's Web site has tables that are not set up properly and contain information about the items on the left and their prices on the right, which cannot be viewed simultaneously, then the user will have to repeatedly scroll right to see the price of the item. This can make the customer impatient and make him/her choose to leave the Web site.

In the above example, it would be more convenient if the price is placed closer to the item. Also, you need to verify whether the columns are wide enough. Check if every row has a wrap-around to make contents clear to the user and if there are any rows that have become too lengthy because of an entry.

8. ***Wrap-around:*** When you have images on your Web site, ensure the correct wrap-around of those images. If the text in your Web site refers to a picture on the right, ensure the availability of the same. Make sure that your Web site does not contain any widow or orphan sentences and paragraphs due to presence of images.

Example: If a Web site explains the concept of water fall model of software testing with water fall model figure, it should include the figure either above, below, or next to the text. In the absence of figure, the text becomes widow or orphan.

Notes Wrap-around of text is nothing but placing the text above, below, or next to an image.

9. ***Functionality:*** If the links in a Web page, database connection, and forms work properly, your Web site functions well. Therefore, in a Web site, you need to test all the links, database connection, and forms used to acquire information from the users.

10. ***Links:*** A link acts as a vehicle that navigates the user from one page to another. Links can be tested in two ways:

    (a) For the existence of the link that navigates a user to another page

    (b) For the existence of the pages to which the existed link is trying to navigate

    You need to ensure that there are no broken links in your Web site.

11. ***Forms:*** When using forms in your Web site, make sure they work properly. When a user fills up forms to submit information, the information should be accepted by the Web site and the submit button should work properly.

Example: If the form is for an online registration, the user should be given login information (that works) after successful registration. When a form is used to gather shipping information, it should be organized well, so that the customer provides all the information required for the shipment.

In order to test the forms, you need to verify that the server is capable of storing the information and that the systems down the line can interpret and use that information.

12. ***Data Verification:*** The data entered as user input for any system should match the business rules that have been defined for the system.

Example: When you are creating a new account registration, the data entered for the phone number (mobile) should consist of ten (10) digits.

13. ***Cookies:*** If your Web site includes cookies, you need to check them also. When using cookies for storing login information, make sure the cookies work and that the information is encrypted in the cookie file. When using cookies for storing statistical data, verify that the totals are counted properly. Also, ensure that those cookies are encrypted, so that they cannot be edited or modified.
14. ***Application Specific Functional Requirements:*** When testing a Web site, you need to verify the application-specific functional requirements, before the user uses it.

Example: While testing a Web site for a shopping center, test whether you can place an order, change an order, cancel an order, enquire the status of the order, change shipping information before an order is shipped, and pay online. You need to ensure that a user can do what is advertised on the Web site.

15. ***Server Side Interface***: Many a times, you may provide links which call external servers for additional data, verification of data, or fulfillment of orders. Therefore, you need to perform the server side interface test as well.
16. ***Server Interface:*** Test whether the browser is interfacing properly with the server.

Example: A bank's Web site can be tested by attempting transactions and then verifying server logs to ensure that what is seen in the browser is actually happening on the server.

It is also a good practice to run queries on the database to ensure that the transaction data is being stored properly.

17. ***External Interfaces:*** Some Web systems use external interfaces.

Example: A shop keeper might verify credit card transactions in real-time to avoid frauds. This case uses an external interface, that is, the shop keeper's Web site approaches the bank's Web site for more information on the credit card used for any transaction.

Some Web sites send several test transactions using the Web interface. While testing for external interfaces, try to test the validity of the credit card. If a shop keeper accepts only Visa and MasterCard, try using an American Express card. The purpose of testing external interface is to ensure that the software handles every possible message returned by the external server.

18. ***Error Handling:*** This is the most often neglected area and hence left untested. Usually, when a Web site is designed, we give importance to error handling mechanism for our system and not errors from other systems or unexpected errors. Hence it is important to check what happens if the site is interrupted in the middle of a transaction.
19. ***Client Side Compatibility:*** It has to be ensured that the web application works on all client machines. To make the Web site accessible to users across the world, the Web site has to be tested with different combinations of operating systems, browsers, video settings, and modem speed, which the users across the world may use to access the Web site.
20. ***Operating Systems:*** Some operating systems may use fonts that are unavailable on other operating systems. Hence, make sure that the secondary fonts are available.

Example: Some fonts are not available on both MAC and IBM compatibles.

As the users may use different operating systems, ensure that the Web site does not use plug-ins available for only one operating system, but uses the ones available for all the operating systems.

21. ***Browsers:*** Check whether the site works with Netscape, Internet Explorer, Linux, and other browsers. Only some browsers make use of HTML scripts. Make sure the availability of alternative tags for images, in case someone uses a text browser. If Secure Sockets Layer (SSL) security is used, it has to be checked for the use of higher browser versions. Also, there should be a message for those using the older browsers.

Did you know? Many sites such as bank Web sites use Secure Sockets Layer (SSL) to increase security on transactions like online funds transfer.

22. ***Video Settings:*** Video settings can be tested using different display sizes such as 640x400 or 600x800 to be appealing to the audiences. Check whether the fonts are too small or too big to read. If the video contains any text and graphic, ensure that the text is working correctly and the graphic is aligned correctly.

Example: When you play a video song, some sites provide the lyrics along with the video. The lyrics have to be displayed along with the song.

23. ***Modem/Connection Speeds:*** Test whether a page takes the standard time bytes per second to load. It is also important to test the loading speed with high-speed connections. It is advisable to make the homepage load as fast as possible. Though users intending to download any software would wait, the same cannot be expected for the homepage.

24. ***Printers:*** Some users would like to read Web content on paper rather than on screen. Hence the content on the Web page should be print friendly. Therefore, you need to check whether the pages print properly. Sometimes images and text align differently on the screen and on the printed page.

25. ***Combinations:*** Try different combinations of video settings and browsers.

Example: Image resolution of 600x800 may look good on the MAC but not on IBM computer. Netscape may work fine with IBM computer but not with Linux.

26. ***Performance Testing***: Performance of the application has to be verified to make sure that the system can handle maximum stress and load for a long period of continuous use. You need to test for the expected level of performance on the client side.

Example: Check for the speed with which the Web pages should appear, and the speed with which the animations, applets, and so on should load and run.

Also, check for the performance of the system during a down time for server and after content maintenance/upgrades.

27. ***Concurrent Users (at the Same Time):*** In some situations, many users may try to access a Web site at the same time. So Web sites should have the feature of allowing many users to access information simultaneously.

Example: When examination results are announced, many students will simultaneously access the same site.

A load test tool would be able to handle concurrent users accessing the site at the same time.

Load on a server can be determined by the number of hits per unit time. The expected performance under such loads such as Web server response time and database query response time are determined by the load test tool. The tools that can be used to test the performance of a Web site include Web load testing tools, Web robot downloading tools, and so on.

28. ***Large Amount of Data from Each User:*** Test whether the site is able to handle large amounts of data from the users.

Example: What if a university bookstore decides to order 6000 copies of a book? Or what if one user wants to send a gift to many friends on friendship day?

29. ***Security:*** While performing online transactions, security is an important factor. This should hold good even when the transaction is in process. It means that you need to ensure that the site is not hacked during the transaction.

Caution If a page on the Web site of any company is hacked, the customers will feel that it is not safe to do business with that company.

Under any circumstance, customers are important. Always consider the audience, the browsers that they use, and the connection speeds they prefer to use. Identify whether the audiences are intra-organizational or external. If the audiences are intra-organizational, they might go with high connection speeds and similar browsers. If the audiences are Internet-wide users, they would go with a wide variety of connection speeds and browser types.

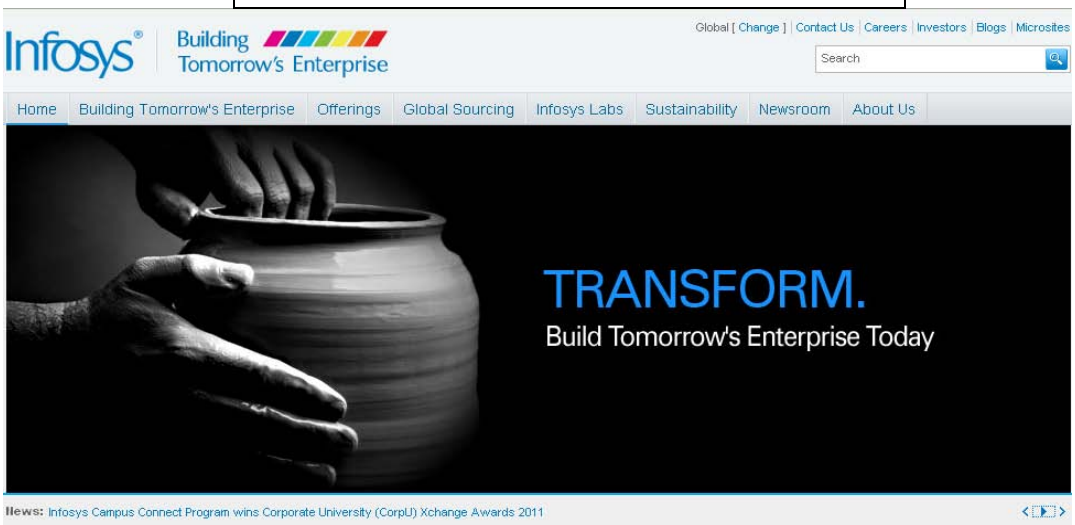
Did you know? There is a myth that performance testing should be performed at the last stage of the development. In reality, it has to be identified and fixed throughout the development lifecycle.

8.2 Black Box Testing

Black box testing is a testing strategy, wherein the tester is not required to have any knowledge of the internal logic, design, or code. Black box testing can be well understood, if you assume the Web page or the entire Web site as a black box. In this strategy, there is no need for a tester to know how a Web site works and what the test specifications are, as the objective is to test the Web site without any predefined inputs.

Figure 8.5 depicts a screen image of Infosys Web site. We can see all the fundamental elements such as texts, graphics, and hyperlinks.

Figure: 8.5: Website with Texts, Hyperlinks, and Graphics



Source: www.infosys.com

Let us now discuss how to test texts, hyperlinks, graphic, and forms using the black box testing methods.

8.2.1 Text

The text of a Web site or a Web page should be treated similar to the text in any document. While testing for the text you should consider the targeted audience level, the terminology used, the depth of content, the subject matter, the accuracy of the information collated, and the routine aspects of spellings, punctuations, and so on.

Why to check for spellings?

When spell checkers are used on Web page content, it may not check the entire text. The spell checkers may only check the regular text and not what is provided within the graphics, scrolling marquees, drop down lists, forms, and so on. Hence it is required to manually check for spelling errors.

Web pages may include contact information such as email addresses, phone numbers, or postal addresses to ensure that these are correct. The copyright notices should be acknowledged correctly and dated appropriately as mentioned. Check for the correct title for all the pages. The title of the page can be seen in the title bar of the browser (upper-left corner of Figure 8.2) and what is listed by default, when you add the page to your favorites or bookmarks.

8.2.2 Hyperlinks

A Web site includes hyperlinks that can be linked to text or graphics. Each link should be tested to make sure that it takes the user to the correct destination and opens in the same tab or in a new window. If there is no specification for the Web site, test if the link is working correctly.

Make sure that hyperlinks are noticeable, text links are underlined, and the mouse pointer icon change (usually to a hand pointer icon) when the mouse pointer is placed on any of the hyperlinks. Verify all the links that are present in the sitemap.

8.2.3 Graphics

Black box strategy for Web site testing includes checking a few obvious things about the graphics. Check if all the graphics are loaded and display properly. If a graphic representation is missing or incorrectly named, it will not load and the Web page will display an error where the graphic representation was meant to be present.

If a Web page has both text and graphics intermixed, the tester has to ensure correct wrapping of the text around the graphics. A tester can try resizing the browser's window to test if there is any incorrect wrapping around the graphic.

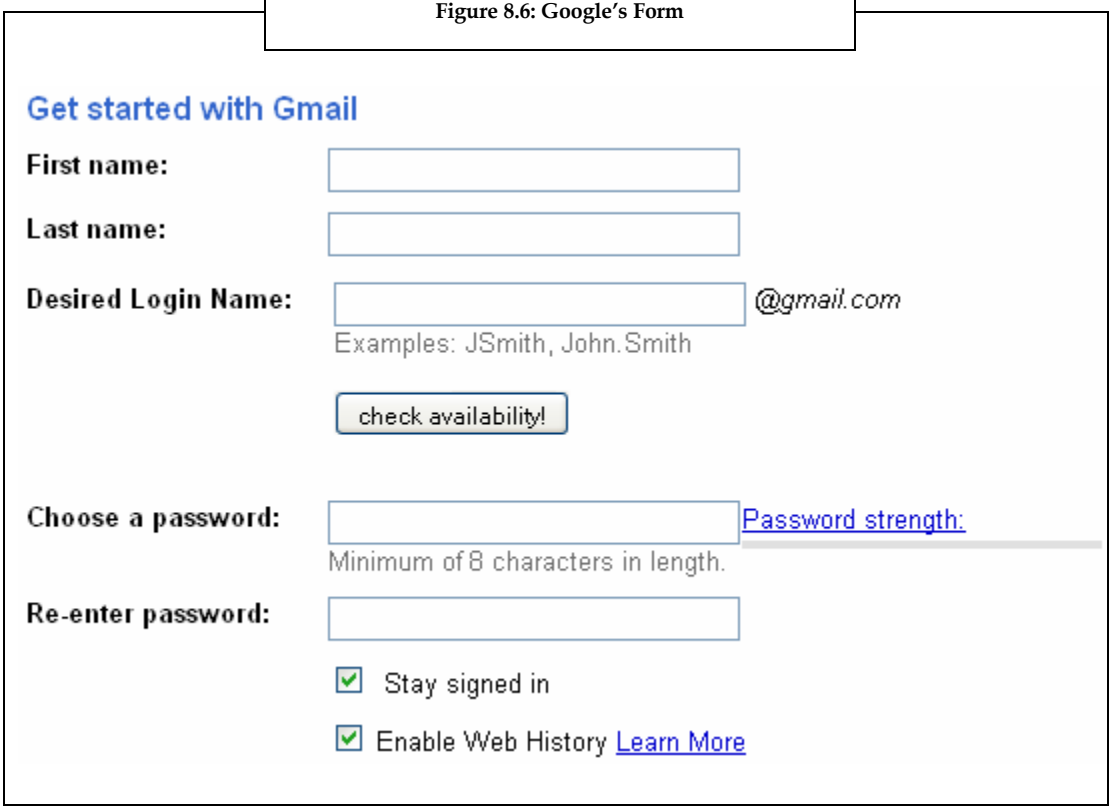
If there are many graphics on a page, it may take a lot of downloading time, which may also lower the Web site's performance.

8.2.4 Forms

Forms are text boxes, list boxes, and other fields that allow a user to enter or select information on a Web page.

Figure 8.6 depicts a form from Google's Web site. It is a sign up form for creating a Google account. This form asks you to enter your first name, last name, desired login name, and password.

Figure 8.6: Google's Form



Source::
(https://www.google.com/accounts/NewAccount?service=mail&continue=http://mail.google.com/mail/e-11-33ebf0ccb45412d145020cfbceb912-232e4049696fabc678becf3aa3a94284d5a9048e&type=2)

When the user enters all the required information and submits, the user should obtain the login information to proceed further.


Task

When we use a bank's Web site for online transaction, it asks for some details. Which among those details can be considered as test cases for black box testing?

8.3 White Box Testing and Gray Box Testing

White box testing is a testing technique in which a tester should know the internal design/working of the system being tested. Gray box testing is a mix combination of black box and white box testing systems.

Now let us discuss white box and gray box testing individually.

White Box Testing

White box testing requires a tester to know the internal working of the system being tested. White box testing uses specific knowledge of the programming code to test the outputs, where as in black box testing there is no need to know the internal working of the system. White box testing becomes successful and effective only if the tester has prior knowledge of the output expected from the system. The tester can then verify if the system deviates from its intended goal.

Some of the elements which can be tested during white box testing are explained below:

1. ***Dynamic Content:*** Dynamic content is nothing but graphics and text that varies depending on some conditions.

Example: Time of the day, the user's preferences, or user specific actions.

The developers may use a simple scripting language such as JavaScript for the content in the Web page and embed within the HTML. This is known as client-side programming. For efficiency, most dynamic content programming is placed on the Web site's server. It is called as server side programming, and it would require the tester to have access to the web server to view the code. The dynamic content created should be checked if it is as per the designer's expectations.

2. ***Database-driven Web pages:*** The Web pages that display catalogs or inventories are database-driven.

Example: E-commerce Web pages

For database-driven Web pages, the HTML provides a simple layout for the Web content. Subsequently, pictures, text descriptions, pricing information, and so on are taken from a database on the Web site's server and plugged into the pages. A tester would have to check if the content from the database is correct and relevant to the columns of display.

Notes

Sources of data for database-driven Web pages are:

1. MySQL
2. Interbase
3. Microsoft SQL Server
4. Microsoft Access
5. Oracle
6. Other databases
7. XML Web services

3. ***Programmatically Created Web pages:*** Many Web pages, especially those with dynamic content, are programmatically generated. To create these pages, a Web page designer may type entries in a database and drag and drop elements in a layout program, press a button, and generate the HTML that displays a Web page. If you are testing a Web site that contains programmatically created Web pages, you should check that the HTML creates it as per the designer's expectations.

4. ***Server Performance and Loading:*** Popular Web sites might receive millions of hits per day. Some people may download data from the Web site's server through the computer's browser and some may simply browse for reading the information. If you want to test a Web site for performance and loading, you need to find out a way to simulate millions of connections and downloads. An automated load and stress testing tool like HP's quality center would help you simulate multiple connections.

5. ***Security:*** As we have discussed in the previous section, Web site security issues should always be given much importance. This is because hackers keep trying new and different ways to gain access to a Web site's internal data. Financial, medical, social, and other Web sites that contain personal/confidential data are especially at risk and require closer knowledge of server technology to test them for proper security.

Gray Box Testing

This testing strategy is a combination of both white box and black box testing. The purpose of Gray box Web site testing is to isolate defects related to bad design or bad implementation of the Web site. In gray box testing, the test engineers should have the knowledge or understanding of the Web site and should be able to design test cases or test data based on their knowledge about the Web site.

Example: Consider a case wherein you need to test a Web site whose functionality is to take users' personal details like email id/address and field of interest on the Web form and submit.

The server will get these personal details and based on the field of interest, will pick some articles and mail them to the user's email id/address. The validation of the email will happen at the client side using Java Scripts.

In this case you can test the Web form with valid/invalid email addresses and different fields of interest (similar to the black box testing) to make sure that the Web site is working fine.

However, similar to white box testing, you need to analyze how the Web site can identify invalid mail IDs, how it can restrict sending mail to invalid IDs, and how the server will not receive notice for any failure- messages which have been sent to the client.

While performing Gray box testing for Web sites, following the seven steps given below would prove useful:

1. ***Identifying Threats to the Application:*** The profile of a threat is created to help the testing team to study the application's functions and features in detail.
2. ***Analyzing the Technical Architecture:*** The technical architecture, which is the most important feature in a gray box testing, should be analyzed. The various aspects of authentication, application components, interfacing with external systems, user session tracking and database interfaces are to be studied.
3. ***Analyzing Application Parameters:*** Various application parameters or variables that are used for exchanging information with the web server should be identified and analyzed.
4. ***Mapping Application Parameters to Threats:*** If there are any threats with regard to the above parameters, they are marked for testing. For example, in an internet banking application where funds are transferred from one account to another, the variables such as session ID have to be verified at each transaction point.
5. ***Developing Test Cases:*** Develop the test cases to test both the structural and functional aspects of the Web application.
6. ***Executing Test Cases:*** Use appropriate tools (manual/automated tools) to execute the developed test cases.
7. ***Reporting and Documenting the Results:*** After executing the test cases, report the results. The results are documented to provide metrics information.

Gray box testing is applicable for Web site testing because it can be performed in complex design environment and under inter-operability conditions. The issues which cannot be addressed through black box or white box testing strategies can be easily addressed easily through gray box testing.

Caution Gray box testing does not cover context specific errors of Web applications.

8.4 Configuration and Compatibility Testing

This testing strategy for Web sites is used to test whether a Web site functions properly across different hardware and software environments. The main purpose of this strategy is to check the simple functional acceptance tests or a subset of the task-oriented functional tests on various combinations of software and hardware configurations. Sometimes, it is carried out to create a specific test that considers the error risks associated with configuration differences.

While testing for software compatibility configurations, the following are tested:

1. Versions of the operating system
2. Input/output (I/O) devices extension
3. Network software
4. Concurrent applications
5. Online services
6. Firewalls.

Checking for hardware configurations would check variances in the following:

1. CPU types
2. RAM
3. Graphics cards
4. Video capture cards
5. Audio cards
6. Monitors/Display devices
7. Network cards
8. Methodology in Connection

Apart from the above test issues, the following are also tested during compatibility checking

1. Various font sizes.
2. Browsers with Cascading Style Sheets CSS, JavaScript turned OFF, and pop-up blockers
3. Various screen resolutions and color depths.
4. Various memory sizes and hard drive space.
5. Different network environments.
6. Different printers (Printer-friendly versions)

Test your Web site to verify whether the user can use Web pages adequately in different browsers using different operating systems such as Windows XP, Vista, Linux, and Mac on different hardware platforms. The Web test engineers should consider various versions, configurations, display resolutions, and Internet connection speeds to prevent the Web pages from ending up with awkward bugs.

However, an important point to remember while testing the Web compatibility is that you should first identify the major customer group for whom the Web site is developed. Depending on the user group, you must decide the main browsers and operating systems for testing purpose.

Case Study for White Box Testing

A large merchant organization that was into online business was in the process of developing an online e-commerce Web site. The organization was trying to facilitate its customers to transfer funds to merchant accounts.

The organization had outsourced the payment processing to a third-party firm. The third-party firm came out with payment software that was able to provide secured interfaces to facilitate funds transfer between the customers and the merchant organization.

The Web site was analyzed under a high-level security risk analysis. When the organization performed the risk analysis, it was identified that one of the risks was occurring during the processing of the transaction, that is, between the payments interface and the Web site. The fake transaction occurring was serious to both the customers and the merchant organization. Due to this, the customers could undergo financial loss, that is, the account balances could get depleted. The fake transaction could damage the credibility and the reputation of the merchant organization.

Using the payments interface, a systematic white box testing was performed on the modules. First, all the module interfaces were determined as interface diagrams. Then, the module interactions were represented with trust relationship boundaries. Finally, data flows among different modules were drawn. Based on this information, some test cases were developed. One of the test cases was to check whether an anonymous user could perform a transaction. The trust relationship mapping and data flow revealed the fact that the system was allowing anonymous users to perform transactions. A path where the system was not validating the user inputs was identified. Then, a test case was developed to test whether an invalid account transfer could take place from an external account to the merchant account. As a result of unauthorized transactions that were occurring through unauthenticated channel, the account transfer was completed successfully.

Risk analysis also noticed a weak authentication channel in the payment customer service component of the Web site. Hence case trust relationship boundaries and data-flow analysis similar to the above situation were drawn for this authenticated channel. After analyzing and testing, it was realized that an attacker could directly gain access to the merchant organization accounts. Using this access, the attacker could make transactions from a merchant account to another non-merchant account.

Because of the bugs, an attacker was funneling the customers' payments to a non-merchant account. The above explained bugs impacted the merchant organization with significant security issues.

From this case study we can conclude that performing white box testing for important modules helps to uncover design assumptions and implementation errors rapidly.

Questions

1. How was the merchant organization able to identify the first bug associated with their Web site?
2. Explain how an attacker was able to perform anonymous transactions.

Adapted from http://basicqafundamentals.blogspot.com/2011/01/case-study-for-white-box-testing.html

8.5 Summary

- Web sites are important for any business to represent itself to the world.
- Web site testing ensures proper functioning of a Web site.
- Home pages, links, and content are the fundamental components of a Web site.
- Links should be tested to ensure the correct functioning of a Web site.
- A Web site should allow concurrent users to simultaneously access the Web site.

- A Web site's text is tested to check whether the text is matching the audience level.
- While testing the hyperlinks, check whether the mouse pointer icon changes when placed on the hyperlink.
- To conduct black box testing for a Web site, the tester need not be aware of the internal design or code.
- To conduct white box testing for a Web site, the tester should have the knowledge of the internal working of the system being tested.
- In white box testing, a tester has to test dynamic content, database-driven Web pages, programmatically created Web pages, server performance and loading, and security.
- Gray box Web site testing identifies the defects due to bad design or bad implementation of the Web site.
- Configuration and compatibility testing is carried out to test the compatibility of the functional acceptance simple tests or a subset of the task-oriented functional tests on various combinations of software and hardware configurations.

8.6 Keywords

Address Bar: An address bar is a text field in a web browser, which displays the Web site address or the Universal resource locator (URL) to the user.

Cascading Style Sheets (CSS): It is a simple mechanism for adding styles such as fonts, colors, and spacing to Web pages.

Cookies: Cookies are a piece of text that is stored on a user's computer by the web browser. There are various reasons for storing them which includes authentication, storing site preferences or shopping cart contents.

Client/Server Applications: This type of architecture works with as a two-tier model where there are two computer programs -- one program, the client, makes a service request from another program, the server, which fulfills the request.

Firewall: It is a device that protects networks from unauthorized access by permitting legitimate communication networks.

SSL (Secure Sockets Layer): It is the standard security technology used to create an encrypted link between a Web server and a browser.

8.7 Self Assessment

1. State whether the following statements are true or false:
   (a) Intranet facilitates users to search worldwide for information on any Web site.
   (b) Home pages include a header at the top which represents the type of the site.
   (c) White box testing is a testing strategy which requires a tester to know the internal design or code.
   (d) Dynamic content is nothing but graphics and text that varies depending on some conditions.
   (e) While testing the Web compatibility, one needs to decide the main browser and OS for testing depending on the testing tools.
   (f) A navigational map helps the user to go straight away to the information which they want.

2. Fill in the blanks:

   (a) For efficiency, most dynamic content programming is placed on the _____.

   (b) Try resizing the _____ window to test if there is any incorrect wrapping around the graphic.

   (c) Load on a server can be determined through the _____.

   (d) The testing that can be performed in inter-operability conditions is _____.

   (e) If you use _____ for storing statistical data, verify that totals are being accounted properly.

3. Select a suitable choice for every question

   (a) Identify the applications that run in Web pages.

      (i) Applets

      (ii) Java scripts

      (iii) Plug-ins

      (iv) CGI scripts

   (b) Identify the applications that run on the server side.

      (i) Database interfaces

      (ii) Java scripts

      (iii) Dynamic page generators

      (iv) Logging applications

   (c) While testing for software compatibility configurations, one must test for:

      (i) Input/output (I/O) devices

      (ii) Extension

      (iii) Connections types

      (iv) RAM

   (d) In a Web site, content can take the form of:

      (i) Internet resources

      (ii) Graphic images

      (iii) Sounds

      (iv) Downloadable movie clips

   (e) Name the testing strategy that is used to test that a Web site functions properly across different hardware and software environments.

      (i) White box testing

      (ii) Black box testing

      (iii) Gray box testing

      (iv) Compatibility testing

8.8 Review Questions

1. "Once the Web site goals have been defined, it is important to have metrics and mechanisms to determine whether the site is providing the defined benefits or not." Mention the metrics used to measure the Web site's performance.
2. "While testing a Web site we need to consider some points". Discuss those points.
3. "A site map and/or navigational map help the user to go straight away to the information which they want". Discuss how a site map can be tested.
4. "In a Web site, patterns and pictures take away the user". Explain how this issue can be addressed.
5. Assume you are using cookies to store some statistical data in your Web site. Briefly explain how you will handle cookies safely.
6. Explain as to why a Web site tester should not depend on spell checkers for checking text.
7. "The graphic or text that varies depending on some conditions" is called as dynamic content. Analyze how a developer can create dynamic content.
8. "Gray box testing is a mix of black box and white box testing". Explain how gray box testing is different from other two testing strategies.
9. "While testing a Web site, the tester has to develop some test cases". Explain the importance of test cases.
10. Even though a Web site is performing satisfactorily with Win XP, testing is recommended. Discuss.
11. Target audience group should be considered while testing a Web site. Justify the statement.
12. Text, graphic, hyperlinks, and forms are the fundamental elements of a Web site. Explain black box testing for these fundamental elements.

Answers: Self Assessment

1. (a) False (b) False (c) True (d) True (e) False
   (f) True
2. (a) Web site's server (b) Browser's (c) Number of hits per unit time
   (d) Gray box testing (e) Cookies
3. (a) Applets, Java scripts, Plug-ins
   (b) Database interfaces, Java scripts, Logging applications
   (c) Input/output (I/O) devices, Extension, RAM
   (d) Graphic images, Sounds, Downloadable movie clips
   (e) Compatibility testing

8.9 Further Readings


Books

Vasudevan V. (2008). Application Security in the ISO27001 Environment: IT Governance publishing.

Mendes.E, & Mosley.N. (2006) Web engineering. Germany: Springer- Verlag Berlin Heidelberg.


Online link

http://ezinearticles.com/?Importance-of-Web-Testing&id=2503273

http://www.softwaretestinggenius.com/articalDetails.php?qry=400

http://sqa.fyicenter.com/FAQ/Software-Testing methodolog/How_to_performance_Compatibility_and_Configurati.html

https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/white-box/259-BSI.html

Unit 9: Automation Testing

CONTENTS

Objectives

After studying this unit, you will be able to:

- Discuss the benefits of automation testing
- Explain the importance of random testing
- Describe bug bashes and beta testing

Introduction

Automation testing is a process carried out using software. Automation testing controls the execution of tests, compares the actual outcomes to predicted outcomes, sets up preconditions and other test controls, and creates test reporting functions. The automation test procedure involves automating a manual process by executing the tests without any manual intervention.

Need for Automation Testing

Software testers are often in need of techniques that make their jobs more effective, and automation testing is one of them. Automation greatly helps in areas where repeated testing needs to be done, especially testing during the release of new version of software. Testers would like to avoid the tedious work of regression testing. Regression testing is performed whenever there are changes made to the functionality of existing software. Patches would have been added or their configuration could have been changed to the software. Regression testing checks to see if the new changes made affects the related modules. This repeated testing can be easily handled through automation testing.

In automated testing, all the tests are performed in the test program. The results of these tests are automatically added to the database. The advantage of recording in a database is that the information about the bugs of the software can be recognized using the statistics available in the database. The best way to increase efficiency, competence, and coverage of software testing is to use the automated software testing.

The automated software testing tools help in playing the pre-recorded and pre-defined actions, comparing the results with the expected behavior, and reporting the success of the manual tests to a test engineer. The automated tests can be repeated, and if required, used to perform tasks not possible with manual testing. This is the reason for automated software testing being an essential component of successful development projects.

9.1 Benefits of Automation Testing

Automation testing uses strategies, tools, and artifacts that reduce the need of manual or human involvement in unskillful, repetitive, or redundant tasks. Automation has greater test coverage, which encourages organizations to test more often and more completely.

The process of automation includes the following:

1. Complete test cases, including predictable results.
2. A standalone test environment, which includes a test database that is restorable to a known constant such that the test cases can be repeated each time there are modifications made to the application.

There are many benefits of test automation. Automated tests reduce the time that is required to test certain conditions. The speed of testing in comparison to manual testing is different – automation testing is fast – it takes an average five seconds to test conditions. Discussed below are a few benefits of automation testing:

1. ***Automated Software Testing Helps in Saving Time and Money:*** It is seen that the software tests have to be repeated on a regular basis during development cycles to ensure quality. Whenever a source code is modified, software tests should be repeated. For every release of the software, it should be tested on all supported operating systems and hardware configurations. It is not possible to repeat these tests manually as it is costly and time consuming. Once the tests are created, automated tests can be run several times at no additional cost and they are much faster than manual tests. Automation testing can be run through the night which saves testing time. Automated software testing helps in reducing the time required to run repetitive tests from days to hours.
2. ***Automated Software Testing Helps in Improving Accuracy:*** Even the most careful tester will commit mistakes during monotonous manual testing. Automated tests perform the similar steps precisely, every time they are executed and always record detailed results.
3. ***Automated Software Testing Helps in Increasing Test Coverage:*** Automated software testing increases the depth and scope of tests that help in improving software quality. The lengthy tests avoided during manual testing can be run unattended. These tests can be performed on multiple computers with diverse configurations. The automated software tests help in executing different complex test cases during every test run, providing coverage not possible with manual tests. With the arrival of automated software testing in the market, the complex features are being dealt with and the testers need not repeat manual tests. This in turn provides more time to create new automated software tests.
4. ***Automated Software Testing Helps in Performing Tasks which Cannot be Performed by Manual Testing:*** It is not possible to execute a controlled web application test with thousands of users. The automated software testing replicates several virtual users interacting with network or web software and applications.

5. ***Automated Software Testing Helps Developers and Testers:*** Developers can use shared automated tests to quickly identify problems before sending them to the Quality Analyst (QA). The tests are executed automatically when a source code changes. The changes are checked and a notification is sent to the team or the developer, if the tests fail. These features help developers save time and increase confidence.
6. ***Automated Software Testing Helps to Improve Team Morale:*** Automation of test processes produces convenient test reporting. These reports aid the testers to spend time on the analysis of the reports and not on generating the test condition. This enables the team to spend time on more challenging and rewarding projects. Team members can develop their skill sets and confidence and, in turn, pass those gains to their organization.

Example: Borland's suite of tools (formerly known as Segue) helps organizations address risks with their suite. Let us know some of the tools.

SilkRadar- Automated defect tracking

SilkPerformer – Automated tool for load and performance testing.

There are many types of scenarios when testing can be automated. These include the following:

1. ***Functional:*** The functional testing process is executed to check whether the software application performs and functions according to the design specifications. In the functionality testing process of a software application, the test engineers analyze the application and develop a complete test plan that covers all the functional areas and features that would be tested. The test plans comprise test cases that demonstrate the testing of each feature and functional area.

Task

Can every aspect of functionality be automated? Consider a banking application where fund transfer is the main activity. What aspects of this application can be automated and what aspects should be manually tested?

2. ***Regression:*** Regression testing is defined as a testing process carried out when a change is made in the software for any reason. This test is performed to check if the software works in the specified way and does not have any negative impact on the functionality provided previously. Regression testing is executed to verify the following:
   (a) The application works as specified after the changes are made.
   (b) The changes made to the application do not introduce any new bugs.

   Regression testing plays an important role in testing the previously tested software. It is an important aspect in various software methodologies where software changes occur regularly.

Example:

In programming methodology, small incremental changes are made to the system based on the end user feedback. Each change requires more regression testing to be performed to ensure correctness with existing functionality.

SilkTest is an automated functional and regression testing tool from Borland.

Similarly, **QuickTest** Professional from HP Quality center is used for E-business functional testing.

3. ***Exception or Negative:*** Negative testing is a testing process in which test methods are designed to see if they are able to handle exceptions. Such methods are considered to be successfully executed when the anticipated exception type is thrown.

4. ***Stress:*** Stress testing is a process in which the software is tested for its effectiveness in providing steady or satisfactory performance under extreme and adverse conditions. These may include heavy network traffic, heavy process load, under or over clocking of underlying hardware, and working under maximum requests for resource utilization of the peripheral or in the system. Stress testing helps to estimate the level of robustness and reliability, even when the limits for normal operation of the system are crossed. Stress testing is considered vital with respect to software that operates in critical or real time situation.

Example: Consider a browser window. Users can open multiple browser windows to navigate between different pages at the same time. However, all these windows are dependent upon one another, and in case one browser crashes, all of them crash. Stress testing is carried out in this scenario to test the browser.

Load2Test is a stress, performance, and load testing tool from Enteros Inc.

5. ***Performance:*** Performance testing is a testing process used to determine the speed or effectiveness of software. This process involves performing quantitative tests in a laboratory such as measuring the response time, the number of instructions executed per second at which a system functions. The qualitative attributes like reliability, scalability, and interoperability are also assessed. Performance testing is often conducted with stress testing. Performance testing is used to confirm that a system meets the specifications stated by its manufacturer. Performance testing can compare two or more devices or programs in terms of factors like speed, data transfer rate, bandwidth, efficiency, or reliability. Performance testing can also be used as an analytical aid in locating communication blocks. It has been noticed that often a system works in a better way if a problem is resolved at a single point or in a single component.

Example: Even the fastest computer would not function properly on today's Web if the connection occurs at only 40 to 50 Kbps (kilobits per second).

6. ***Load:*** Load testing is a process of subjecting a computer, peripheral, server, network, or application to a work level that is approaching the limits of its specifications. The load testing process can be performed under controlled laboratory conditions to compare the capabilities of different systems or to precisely measure the capabilities of a single system. Load testing can also be performed in a field to obtain a qualitative idea of the performance of a system in real world. Load testing is considered as a part of a more general process known as performance testing.

Example: Few examples of load testing are:

(a) Transferring a number of tasks to a printer at a time.

(b) Loading a server with a large amount of e-mail traffic.

(c) Operating multiple applications on a computer server.

The above mentioned testing types can be automated.

In nearly all software development models, the code-test-fix loop gets repeated many times before the software is released. If the tests are carried out for a particular feature, they are to be performed several times. The tests are performed to check if the bugs found previously are fixed and that no new bugs are introduced.

Example: Hewlett Packard offers a suite of solutions called HP Quality center which automates testing and quality assurance for client/server software and systems. They can be used for e-business applications and enterprise resource planning applications. The following are the different products which are part of the suite.

Quality center – This product can be used for tests and defect tracking.

Load Runner – This product helps in enterprise load testing,

WinRunner – This product helps in test automation for the enterprise.

Example: If a software project has many test cases to run, there may be hardly enough time to execute them just once. Running them several times might be impossible. In such a situation, software test tools and automation can help solve the problem by providing a more efficient means to run the tests, when compared to manual testing.

Did you know? Automated testing involves higher upfront costs and should be looked at as a long-term investment, where the pay-offs come anywhere between 2-4 years down the road.

Task

Read on how ROI on Test automation is carried out from the following link

www.keane.com/resources%2Fpdf%2FWhitePapers%2FWP_ROIforTestAutomation.pdf.

9.1.1 Test Tools

While performing automation software testing, a tester comes across various testing tools. The types to be used are based on the type of software that a tester is testing.

The advantage of these test tools is that the tester need not be an expert on how they work or what exactly they do in order to use the test tools.

Example: Consider a tester testing networking software that permits a computer to communicate with up to 1 million other computers at the same time. It can be difficult to carry out a controlled test with 1 million real connections. But, with a special tool that replicates those connections, the tester can adjust the number from one to a million and perform the tests without having to set up a real-world scenario.

It is not necessary to understand how the tool works. The only requirement to test a particular application would be the knowledge of white box skills and awareness of the low level protocol to effectively use this tool.

In this section, we will discuss the major classes and uses of testing tools. Some instances are based on tools included with most programming languages, whereas others are commercial tools that are sold individually. In many cases it is seen that a new tool has to be developed to satisfy the tester's needs.

Viewers and Monitors

A viewer or monitor test tool allows the tester to view details of the software's operation, which otherwise would not have been possible.

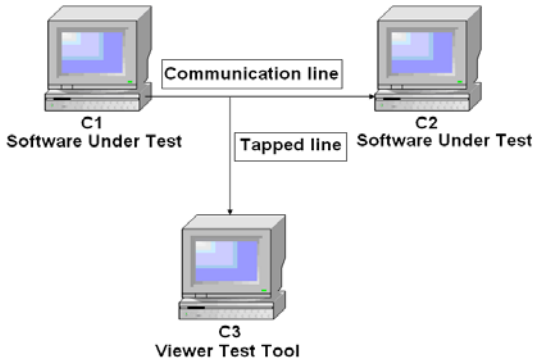
Example: An example of the viewing tool is the coverage analyzer tool.

Most code coverage analyzers are intrusive tools since they need to be compiled and linked into the program to access the information they provide.

A communications analyzer is another type of viewer. This tool enables the tester to see the raw protocol data moving across a network or other communications cable. It taps into the line, pulls off the data as it passes by, and displays it on another computer.

Figure 9.1 illustrates the working of a communication analyzer.

Figure 9.1: A Communications Analyzer



Here, a user can enter a test case on C1, confirm that the resulting communications data is correct on C3, and then check whether the appropriate results occurred on C2. A user can also use this system to investigate why a bug occurs. By looking at the data pulled off the wire, the user can determine whether the problem occurred in creating the data (C1) or interpreting the data (C2).

The code debuggers are also known as viewers since they allow programmers or white-box testers to view internal variable values and program states.

A viewer test tool is classified as a tool which enables a user to look at data that is not visible to an average user.

Drivers

Drivers are tools used to control and operate the software being tested.

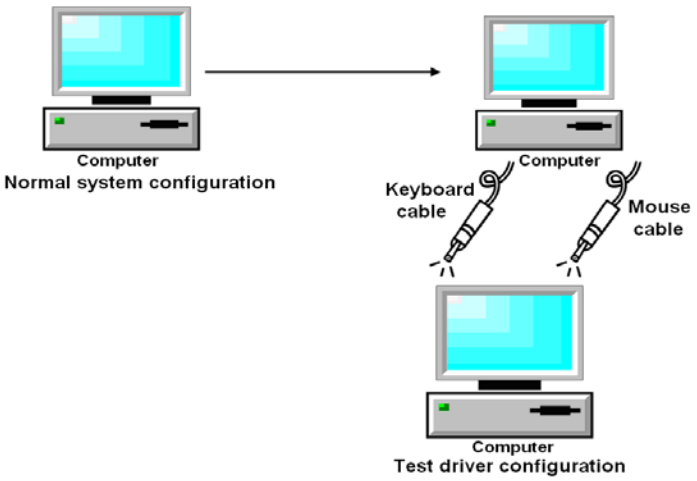
Example: A batch file or a simple list of programs or commands executed sequentially, is an example of a driver tool.

In the earlier days, MS-DOS was a popular means for testers to execute their test programs. The testers would create a batch file containing the name of the test program and would start running the batch of programs. But with the upcoming new operating systems and programming languages, there are many sophisticated methods of executing test programs.

Example: Java or a Perl script can replace an old MS-DOS batch file, and the Windows Task Scheduler can be used to execute various test programs during any time of the day.

Figure 9.2 depicts a computer that acts as a driver test tool.

Figure 9.2: A Computer that Acts as a Driver Test Tool to Replace the Keyboard and Mouse of a System Being Tested.



Here, a computer is being depicted as a driver test tool that can replace the keyboard and mouse of a system being tested. Assume that the software being tested requires large amounts of data to be entered in the test cases. If there are a few hardware modifications and a small number of software tools, the user can replace the keyboard and mouse of the system being tested with an additional computer that can operate as a driver. With this driver computer, one can write simple programs that automatically generate the appropriate keystrokes and mouse movements to test the software.

Alternatively, a user can also opt for another method, that is, a user can simply run a program on the first system that sends keystrokes to the software being tested. But again there are two problems involved in this. They are:

1. If the software or the operating system is not multitasking, it is not possible to run another driver program at the same time.
2. If the keystrokes and mouse movements are sent from an external computer, the test system is non-invasive. If a driver program is being operated on the same system as the software being tested, it is invasive and may not be considered as an acceptable test scenario.

Stubs

Stubs are used to receive or resend data that the software sends. They do not control or operate the software being tested. For example,

Example: If a user wants to test software that sends data to a printer, one way to test it is to enter data, print it, and check the result. This is a slow , incompetent, and error prone process. It is also difficult to obtain the appropriate results as the user may not know if the output had a single missing pixel or if the color was different. Here, accurate results can be obtained if the user replaces the printer with another computer operating on stub software.

Stubs are often used when the software needs to communicate with external devices. They make the testing process efficient as they permit testing despite lacking the hardware.

Stress and Load Tools

Stress and load tools are used to test extreme conditions on software like the maximum number of concurrent users or maximum number of transactions that could be performed. If an application is being executed on the system with all available memory and disk space, it may work fine. But, if the system does not have a few of these resources, there is a possibility of the occurrence of a new bug. A user can copy files to fill up the disk, execute many programs, and perform various tasks, but these methods are ineffective. This is the reason why a stress tool is introduced. The stress tool makes the testing much easier. The stress program permits a user to set the values individually and use other resources available to the software running on the machine.

Interference Injectors and Noise Generators

Interference injectors and noise generators are another class of tools. These tools are similar to stress and load tools, but are more random in their functions.

Example: The stress utility has an executor mode that changes the available resources randomly. A program can be executed properly with the available memory, but if the available memory changes constantly, it may create a problem. The executor mode of the stress utility will not be able to discover these types of bugs.

Analysis Tools

Analysis tool is the most commonly used tool. Following are the tools used by software testers to simplify their work. These tools help in getting the job done and are less time consuming.

1. Word processing software.
2. Spreadsheet software.
3. Database software.
4. File comparison software.
5. Screen capture and comparison software.
6. Debugger.
7. Binary-hex calculator.
8. Stopwatch.
9. VCR or camera.

With the constant advent of new technology, it is necessary to select an appropriate tool to check the complication and direction of the software on a regular basis.

9.1.2 Software Test Automation

Software test automation is a class of software testing tools. The software test tools mentioned above are effective, but these tools must be operated or monitored manually. These tools can be formed and run as a test suite, and run with little or no involvement from the tester. This results in a faster test process that involves looking for bugs, analyzing them, and logging the results. In this section, we will study the different types of automation, progressing from the simplest to the most complex.

Macro Recording and Playback

Macro recording and playback is known as the basic type of test automation. This tool is used to record the keyboard and mouse actions as the tester runs the tests for the first time and then plays them back when required to run again. Recording and playing the macros again is a simple task when the tests are performed for Windows or Mac.

Example: On Mac you can use Quick Keys; and on Windows the shareware program, Macro Magic can be used.

A huge number of macro record and playback programs are available in the market, so you can select the one which fits your needs. Macro recorders and players are a type of driver tool. We know that drivers are the tools used to control and operate the software being tested. The same process is performed when working with macros. The recorded macros are played back, repeating the actions that were performed to test the software.

Macros can be effectively used to perform automation testing, making the work of a tester easy and fast. However, they are not considered perfect for testing, the reason being the lack of verification. The macros are unable to check the purpose of the software. Though this is a serious issue, macros are still a favorite tool for a few testers. This is because macros can get rid of all the repetitive typing and mouse moving. Hence, the tester just needs to run the macros and substantiate that the results are as expected. Another issue with macros is the playback speed. Though the speed of the playback can be adjusted, it is quite difficult to keep the macros coordinated.

Example: Let us assume that a web page takes 20 seconds to load. The speed of the macros can be adjusted to keep a track of the expected worst case. This could give rise to a problem, that is, the software may run fast and the macros would be slow. If a situation arises, wherein the page takes more time to get loaded (than the expected time), the macros would end up clicking the wrong things at the wrong time.

Programmed Macros

Programmed macros were designed to provide a better feature than the simple record and playback variety. It is advisable to create macros using simple instructions for the playback system to follow, rather than creating a programmed macro by recording the actions. The benefit of using programmed macros over macro recording and playback is that the programmed macros can save time as they do not depend on absolute delays but wait for certain conditions to occur before they go on. The macro language is simple and easy to use. It also provides generic commands which prompt the tester for information. However, following are the few important points to be considered for programmed macros:

1. Programmed macros can only loop and repeat
2. Variables and decision statements are not available
3. The tester cannot check the results automatically

Hence, a more advanced automated testing tool is required for the testing process

9.2 Random Testing

Random testing is a type of functional testing. This form of testing is used when the problem is complex and it is difficult to test every combination. The release criteria can include a statement that shows the amount of random testing required. There are a few random testing techniques applicable for a project. Every project has its own criteria and accordingly a random testing technique is used. But the testing techniques are mentioned in the test plan.

Mentioned below are the main types of random testing techniques:

1. ***Random Input Data Generation:*** This testing is carried out when a new functionality is added to an application. For testing the same, the tester will randomly generate date for all existing and new fields in the application being tested.

Example: **GenerateData** is a free open source script that helps to generate large volumes of custom data. It has been written in JavaScript, PHP and MySQL.

DBMonster helps to generate random test data and inserts it into SQL database.

2. ***Random Sequence of Data Input:*** This type of random testing is also known as stochastic testing.

Example: There is a GUI interface with multiple tabs for entering information and only one save button. The system will work correctly if the order of entering information is tab 1, tab 2, and tab 3, but may not work properly if the sequence is tab 2, tab1, and tab3.

3. ***Random Data Collection from Existing Database:*** This is another type of random testing technique.

Example: If a tester wants to test the functionality of an application, the tester would randomly select members from existing system database and verify that the new functionality works as expected.

Given below are a few types of algorithm used in random testing:

1. Uniform random walks in which every trace may have the similar probability to occur.
2. Randomized quasi-random sequences.
3. A path-oriented random test data generator.
4. Sequence of random test data that can perform only a subset of paths in a program.

The random testing inputs will be able to provide proper results only if:

1. Correct algorithm is used.
2. Inputs are distributed evenly over the input domain.
3. The risk factors are considered during the distribution of inputs.

Random testing is not only considered as a useful testing technique in itself, but it can also be used in many other testing techniques. Also, random testing has high failure detection effectiveness during GUI testing.

Another method of testing is the adaptive random testing. Adaptive Random Testing (ART) indicates a family of testing algorithms with better performance compared to pure random testing and with respect to the number of test cases essential to detect the first failure. This testing technique is also known as the Black Box testing technique and is based on the assumption that random testing failure detection efficiency can be improved by evenly distributing test cases in the input domain.

Example: **When selecting the tool to be used to generate random numbers, the tester should keep in mind:**

1. The type of data required
2. The range needed (minimum to maximum)
3. The way of distribution of the data

A log file should exist to store the seed value before using it. This helps in reproducing the error, in case the computer or application fails radically. The quality of random testing can be verified using a code coverage tool. Every random test generator is expected to identify a different set of bug.

Caution Prior to using the random test generator for testing, the tester must open a log file and verify that the data generated is according to the tester's needs.

Random testing has gained immense popularity in the field of gaming and protocol testing. This is because random testing has the power of running a large number of test cases with high failure detection effectiveness. Though we do not acquire perfect or optimal results with random testing, we can obtain good results at a low cost. In some circumstances, random testing methods are considered

more practical than any other alternative method. The coverage measures allow an individual to understand the problems of a selected algorithm for random testing, and hence provide opportunities for future improvement.

Example: TestIT! (from TdgTeam) is a powerful testing tool that generates great quantity of random but real-life data.


Notes

When testers perform random testing, they must ensure that the tests are adequately random.

9.2.1 Gorilla Testing

Gorilla Testing is a testing technique used to verify defensive programming. Defensive programming refers to a programming method wherein the incorrect inputs that are expected to be given by the user are predicted, and thus the program is designed to put up with such inputs. Gorilla testing is highly recommended for testing of gaming software.

Example: Assume that a game is given to a small child who is unaware of how to play the game. The child would press the keys arbitrarily; that is, incorrect input is given. In such cases, the software must respond properly. Hence, gorilla testing is used.

It is utilized to check user interfaces in application software for people who are unaware of software products and their usage. Gorilla testing is an easy testing process. The tester has to merely press the keys and check if the software succeeds. In case, an incorrect input is given, then a dialog box appears on the screen. If the error message does not appear on the screen, then the error handling routines have to be improved. But if the system hangs, then it is a major defect that needs to be rectified.

Did you know? Justin Forrester and Barton Miller of University of Wisconsin developed a random testing tool named Fuzz. This tool has the capacity to send valid and invalid Win32 messages or random valid and invalid keyboard and mouse events to a Windows application to carry out Gorilla testing on the application. If the installed software is reliable, the system would not hang or crash. However, it was found that Access 2000, PowerPoint 2000 and Word 2000 failed this test!

9.2.2 Monkey Testing

Monkey testing is yet another type of random testing technique. It is used with a fully automated testing tool. Whenever this testing is performed, it executes mouse clicks on the screen or keystrokes on the keyboard randomly. The monkey testing technique comes under the category of black-box testing. Following are the different types of monkey testing techniques:

1. ***Dumb Monkey Testing:*** Dumb monkey testing is the initial and the lowest level of testing service. This technique can be run by anyone who has limited or no knowledge of testing. This simply means that the testing process is not pre planned. Following are the services provided by the dumb monkey software testing:
   (a) It estimates the required time and staff for testing.
   (b) It does not require pre- planning of the test process and acceptance criteria.
   (c) It executes tests by clicking through an application or software.
   (d) It provides quick results.

2. ***Semi-Smart Monkey Testing:*** The next stage of the dumb monkey testing technique is the semi-smart monkey testing technique. This testing technique is performed by individuals who have basic knowledge in testing. Semi-smart testing technique is successful when it is performed in small groups. The services provided by the semi-smart monkey testing technique are as follows:

   (a) It is the initial level of planning and acceptance criteria.

   (b) It estimates the required staff and time for testing process.

   (c) It interprets the requirements prior to running the test.

   (d) It makes a note of the semi-technical bug reports and also logs the recording process.

   (e) It is also known as the initial level of regression testing.

3. ***Smart Monkey Testing:*** Smart monkey testing is the third type of monkey testing. It is commonly used by testers because of its efficiency. Using the smart monkey testing technique, it is possible to confirm vast testing projects, make use of different testing tools, and also use the initial testing theory. This testing technique is restricted to the testing environment. The smart monkey testing technique is considered to be valuable for stress and load testing as it helps to detect a significant number of bugs. However, the only area of concern is that it is very expensive to develop this testing technique. The following services are provided by the smart monkey testing technique:

   (a) It helps to track systems usage.

   (b) It scrutinizes the requirements prior to running the test.

   (c) It creates circumstances for the test.

   (d) It helps to track the logs of systems usage.

   (e) It also locates memory or resource bugs.

   (f) Monkey testing is a valuable testing technique, but not the only testing technique.

9.3 Bug Bashes and Beta Testing

In the earlier sections, we studied the various tools that can be used for testing particular software. We also learnt about the different testing techniques. In this section we will discuss the other testing methods.

Bug bash is a tool which is used as a part of the test management program. In this method, all the developers, testers, program managers, usability researchers, designers and several others are involved in analyzing the product for bugs. In the bug bash activity, a particular area is selected for testing. The area to be selected can be free from any bugs or it can be the one causing problems during the application. The bug bash can determine the bugs which otherwise could not be determined by normal testing methods. There are many criteria for selecting the area to undergo the bug bash activity.

9.3.1 Test Sharing

Many testers are involved in testing software. All testers will have their own methods and criteria to locate bugs. The testers have an option of swapping their test responsibilities. This allows both the testers to analyze the software properly and also complete the testing task. It is also possible that a tester is unaware of particular software, so by swapping the testing task they can learn the software and also come up with new tests to detect bugs. The testers can also ask the experienced testers for help. Another way to get help is by scheduling the bug bash. Given below are a few tips to organize a good bug bash activity:

1. ***Involving the Entire Team in the Bug Bash Activity:*** It is a known fact that the testers would test the application several times to locate bugs. But if a set of other people, such as designers, business analysts, testers, developers, release managers, user experience designers, sustained engineering teams, and even high level managers are involved in the bug bash activity, then it would be easier to locate bugs if any. Bugs found during the bug bash activity can be a feedback in itself. This feedback helps in making proper decisions in releasing the software into the market.

2. ***Giving High Quality Guidance to People Participating in the Bug Bash Activity:*** While including other people in the bug bash activity, if it is certain that they are looking at the software or application for the first time, it is important to note that they execute the right tests while running the bug bash activity. Given below are a few considerations to be made:

   (a) ***Preparing a High Quality Test Environment for Testing:*** It is necessary to check that the system they are working on does not crash or that they do not face any problems while logging into the system. It is always advisable to maintain different log in accounts record all information in a document and make sure the information has been distributed to all the people involved in the bug bash.

   (b) ***Preparing a Document to Record all Information of the System:*** It is necessary that the testers know how to start the process. Hence, if a document given to them depicts the core scenarios and explains how to execute them, it would be extremely helpful to the testers. By performing this activity, the testers will be able to start in the right direction and then perform test around those scenarios and locate bugs.

   (c) ***Maintaining a Bug Bash Template:*** It is a known fact that many people classify bugs for the first time. Hence the bug management software that the team uses should be free from problems for the new testers. For this reason, it is important to have a template that would ensure that the focus is on locating bugs rather than spending time on the tools.

3. ***Timing the Bug Bash:*** It is necessary to time the bug bash activity. One should time the activity such that it is neither too early nor too late. If it is timed early, then the attention moves away from recording good quality bugs and customer issues, but if it is planned later, it may not be possible to detect the issues in the core scenario flow. The best time to organize a bug bash activity is at the end of the milestone/cycle.

The testers can approach their own team members or the project development team for the test sharing activity.

9.3.2 Beta Testing

Beta testing is known as an external method of testing. In the beta testing process, the software is launched for a selected group of customers who use the software in the real world. This method of making sure that the software has been verified by the end users and validating it is a common method known as beta testing. This testing process usually occurs after the product is developed and is awaiting the confirmation that the software is ready for use.

Following are the assumptions prior to planning a beta test:

1. ***It is Important to Decide Who Performs the Beta Test:*** Consider a tester who wants to test particular software for any remaining usability bug. It is possible that the beta tester is anxious only about the low level operation. Hence, a tester must specify the type of beta tester required for the testing process.

2. ***It is Necessary to Know Whether the Beta Testers Use the Software:*** There may be few cases where a problem was not reported by users or there were no problems related to the software. It is also likely that the beta testers use the product for a limited time. Hence it is significant to check that the software is carefully and properly used by the beta tester and they also report for any kind of bugs.

3. ***Beta Test is Crucial for Determining the Compatibility and Configuration Bugs:*** It is a known fact that it is difficult to recognize and test a sample of the software or hardware in real world. Hence the beta participants must be selected in such a way that they represent the target users and also help in finding the compatibility and configuration bugs for the users.

4. ***Generally Beta Tests are Not Considered to Locate Functionality Bugs:*** The reason is that the beta test occurs at the end of the software development process and if at all any bugs are found, there would not be much time available to fix the bugs.

5. ***It is Necessary to Find Quick-fixers or Innovative Solutions for Bugs:*** Beta testers, along with development team, have to find quick fixes or innovative solutions that help the beta customers in the event of critical bugs.


Notes

Making proper arrangements for the beta test is important. Also, the beta tests must be properly defined and managed.

9.3.3 Outsourcing Testing

It is seen that various companies outsource a part of the testing to other firms who are well versed with the software testing techniques. Though this method can be expensive, it can prove to be very effective when it comes to sharing the testing work. The configuration and compatibility testing could be outsourced, since it usually requires a huge test lab and many people to manage it.


Example: Localization testing is another reason for outsourcing. This is done due to the fact that the testers must be aware of different languages as the product must often support different languages.

Following are a few points to be noted when outsourcing the testing activity:

1. The tasks to be performed by the testing company
2. The schedule to be followed by them
3. The deliverables to be provided to the testing company
4. The deliverables the testing company would provide
5. The type of communication required
6. The expectations that are to be met


Notes

The points mentioned might not be of much importance, but it is advisable that the firm does not overlook these matters**.**


Case Study

Automating Testing for a Leading Insurance Company in Europe and the United States

A large merchant organization that was into online business was in the process of developing an online e-commerce Web site. The organization was trying to enable its customers to transfer funds to merchant accounts.

This case study is for a client of Infosys. The client is a Europe-based financial institution. The institution caters to a million clients in over 50 countries including 14 million in the US. Retirement services, annuities, life insurance benefits to employees, mutual funds and financial planning are the services provided by the institution to the retail and institutional clients. The company also provides financial products and services in insurance, asset management and direct banking.

After a methodical QA process, an automation suite for regression testing was suggested by Infosys. The institution conducted a proof of concept (POC) to assess the feasibility of automating the application and test cases using Infosys' Test Automation Accelerator (ITAA) method. The POC study disclosed that the ITAA usage had resulted in a 20%-30% savings. Infosys built the automation suite using the Global Delivery Model. With the presence of testing professionals and automation experts, the project was successfully completed within time and budget.

Contd..

The result was that the company benefited in a number of ways. Few of them are as follows:

1. The client was able to save a huge amount of money, about 54%.
2. Due to the highly scalable, easy to maintain and time saving method of automation in testing, the client was able to complete the project successfully.
3. A ninety-day warranty support was provided after the sign-off.
4. Production time was reduced.
5. The institution was able to identify new functional test scenarios for regression.

Question

1. Discuss about Infosys Test Automation Accelerator.

Source: http://www.infosys.com/offerings/industries/insurance/case-studies/Pages/automated-testing.aspx

9.4 Summary

- Automation testing is a testing process which is carried out to control the execution of tests, compare the actual outcome to the predicted outcomes, set up preconditions and other test controls, and test reporting functions.
- Automation testing is done using software.
- Automated testing processes are preferred to the manual testing processes. This is due to the fact that manual testing process is sometimes incapable of finding the bugs.
- There are various test tools used by a tester to test the software.
- Software test automation is a class of software testing tools.
- Random testing is a type of functional testing. Random testing method is used by the testers when the problem is complex and it is difficult to test all combinations.
- Random testing consists of two methods. One is gorilla testing and the other is monkey testing.
- Bug bash is a tool which is used as a part of test management program.
- Test sharing is known as an internal method for identifying bugs.
- Beta testing is known as an external method for identifying bugs and is carried out at the last stage of the software development process.

9.5 Keywords

Bug Tester: A beta tester is a person who tests a product before it is released.

Log File: Log Files list actions that have occurred. The listed actions can be analyzed using log analysis tools to get an understanding. Log files are useful for problems analysis and to gather relevant data.

Scalability: Scalability refers to the software application's capability to scale up or scale out. Testing the scalability can identify major workloads and also mitigate bottlenecks.

Seed Value: The seed value is the initialization point. Seeding in a random number will determine the order of the values that is returned.

Interoperability: Interoperability is an attribute that refers to the ability of the software application to work together in diverse systems and organizations.

9.6 Self-Assessment

1. State whether the following statements are true or false:
   (a) Automating testing requires a formal manual testing process which exists in the firm.
   (b) Automated software testing does not help in improving accuracy.

(c) Gorilla testing is considered as an important testing technique in software testing, especially in gaming technology.

(d) It is not important to know who would perform the beta test.

(e) Using smart monkey testing, it is possible to confirm vast testing projects.

(f) Random testing is not a type of functional testing.

(g) Stubs are tools that are used to control and operate the software being tested.

2. Fill in the blanks:

(a) _____ is a process performed by using software.

(b) _____ uses an automatic test program.

(c) Beta testing is known as a _____ method of testing.

(d) _____ technique is mainly performed by the individual who has basic knowledge of testing.

(e) _____ is a type of testing technique which is used to verify defensive programming.

3. Select the suitable choice for every question:

(a) Which among the following testing processes is used to determine the speed or effectiveness of software?

(i) Performance testing

(ii) Load testing

(iii) Stress testing

(iv) Exception testing

(b) Which among the following tools is used to bring stresses and loads to the software being tested?

(i) Stress and load tools

(ii) Stubs

(iii) Drivers

(iv) Viewers and monitors

(c) Which tool allows the tester to view details of the software's operation that otherwise would not have been possible to see normally?

(i) Stubs

(ii) Drivers

(iii) Viewers and monitor

(iv) Interference injector

(d) Which tool is used as a part of test management program?

(i) Bug bash

(ii) Noise generator

(iii) Interference injector

(iv) Driver

(e) Identify the initial and the lowest level of testing service.

(i) Semi-smart monkey testing

(ii) Dumb monkey testing

(iii) Gorilla testing

(iv) Smart monkey testing

9.7 Review Questions

1. "Load testing is a testing process of subjecting a computer, peripherals, server, network or application to a work level that is approaching the limits of its specifications." Explain briefly.
2. "In the beta testing process, the software is launched for a selected group of customers who use the software in the real world." Discuss.
3. "Regression testing is defined as a testing process which is carried out in case the software is modified for any reason." Discuss with example.
4. Assume that you have developed a macro in an application. What would be the pointers when you test the same?
5. "Monkey testing is used with fully automated testing tool." Discuss briefly.
6. "While performing automation software testing, a tester will come across various testing tools." Discuss any two test tools.
7. "The functional testing process is executed to check whether the software application performs and functions according to the design specifications." Explain briefly.
8. "Performance testing is a testing process to determine the speed or effectiveness of software." Justify the statement.
9. "Software test automation is a class of software testing tools." Explain the two types of software test automation.
10. "Gorilla testing is a type of testing technique which is used to verify defensive programming." Discuss with example.
11. "Beta test is crucial for determining the compatibility and configuration bugs." Substantiate
12. "Drivers are tools that are used to control and operate the software being tested." Explain with example.
13. Assume that you are leading a team of testers and you are planning to conduct a bug bash activity. What are the pointers you would observe while scheduling the same?
14. "Random testing has gained immense popularity in the field of gaming and protocol testing" How do you think random testing helps in gaming?

Answers Self Assessment:

1. (a) True (b) False (c) True
   (d) True (e) True (f) False
   (g) False
2. (a) Automated testing (b) Software testing (c) External
   (d) Semi-smart monkey testing (e) Gorilla testing
3. (a) Performance testing (b) Stress and Load tools (c) Viewer and monitor
   (d) Bug bash (e) Dumb monkey testing

9.8 Further Readings

Ron Patton, Software Testing, Second Edition

Elfriede Dustin, Jeff Rashka, John Paul, Automated Software Testing, U.S. A

http://extremesoftwaretesting.com/Techniques/RandomTesting.html

http://www.mactech.com/articles/mactech/Vol.13/13.10/SoftwareTestAutomation/

http://www.automatedqa.com/products/testcomplete/manager-overview/

http://www.calsoftlabs.com/product-testing/functional-testing.html

http://www.exforsys.com/tutorials/testing/what-is-regression-testing.html

http://www.buzzle.com/articles/software-testing-stress-testing.html

http://www.bitpipe.com/tlist/Performance-Testing.html

http://searchsoftwarequality.techtarget.com/definition/performance-testing

Unit 10: Test Planning Fundamentals

CONTENTS

Objectives

After studying this unit, you will be able to:

- Describe the test planning process goals
- Explain test phases in the test planning
- Describe the aspects of strategy in test planning
- Explain the importance of resource requirements
- Explain the testing schedule in test planning
- Describe test cases and the importance of bug reporting
- Explain the importance of test metrics

Introduction

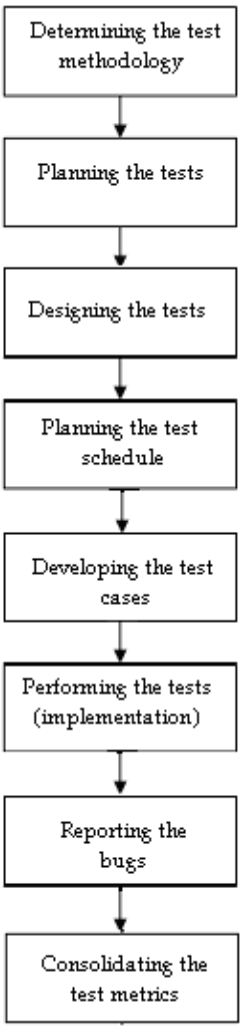
A test plan is the elementary document that you refer while testing any software. The responsibility of creating the complete test plan for a project lies with the test manager. Test planning reveals the entire project's testing schedule and approach. Hence, it is important to know the amount of effort and the kind of information that goes into a test plan.

A good test plan will help to efficiently communicate and document the test effort with well-constructed test plans, test cases, and test reports. It will also help the testers in achieving the desired quality for the product being tested.

10.1 Test Planning

Testing is carried out throughout the software development process. It is important to plan, design, and develop performance metrics to carry out testing. The activities involved in the testing process can be divided into phases, which begin in the design stage and end when the software is installed at the customer's site. The test planning process is illustrated in Figure 10.1.

Figure 10.1 A Typical Test Resource Graph



The main purpose of implementing testing is to test the effectiveness and efficiency of the software. It is an effort to reduce the number of undetected errors present in the system or software being tested.

Despite all measures taken to identify and remove errors, obtaining software that is free of defects is still an unrealizable goal. This challenge requires the testers to maintain high quality of testing in the software. Two ways adopted to improve the testing process are to upgrade the effectiveness of the test cases applied during testing and to develop automatic software testing tools.

Well-constructed test plans, test cases, and test reports help a tester to achieve the goal of correctly communicating and documenting the test activity.

10.2 Goals

The testing process cannot take place without prior communication with the programmers of the software. This is because the testers cannot start testing the software unless they know what the code does and how it works. Similarly, communicating with other software testers is also important. It helps to understand:

1. What to test?
2. What resources will be needed?
3. What will the schedule be?

Hence, without proper communication, the project will have little chance of succeeding. The software test plan is the principal way through which software testers communicate their intent to the code developers.

The IEEE Standard 829 for Software Test Documentation states that the purpose of a software test plan is: "To prescribe the scope, approach, resources, and schedule of the testing activities. To identify the items being tested, the features to be tested, the testing tasks to be performed, the personnel responsible for each task, and the risks associated with the plan."

A test plan is a written document. It is a by-product of the detailed planning process. It describes and summarizes the results of the planning process. This document is also used as reference for future use. The main goal of the test planning process is to communicate and not just record the software test team's intent, its expectations, and its understanding of the testing that is to be performed.


Notes

Points to be considered during testing:

1. An appropriate software quality standard must be ensured.
2. The software testing strategy must be clear

Everything included in a software product need not necessarily be tested. There may be parts of the software which were previously released and have already been tested. Alternatively, the component may have been reused from another software company or a pre-tested component may have been sourced from another company.

It is important to identify each component of the software during the planning process and make known whether it will be tested. There should be a reason for deciding to not test a component. This is because it can be disastrous if a piece of code gets released untested from the development cycle due to any misunderstanding.


Example: A software package for a patient monitoring system in a hospital requires the highest software quality standard, considering the possibility of severe consequences of software failure.

10.3 Test Phases

To plan the test phases, the test team first analyzes the chosen development model and decides whether particular phases or stages of testing should be performed over the course of the project. Typically, in a code-and-fix model, there is only one test phase, that is, test until asked to stop. There can be several test phases in the waterfall and spiral models, which may begin from examining the product specification and continue till acceptance testing. Test planning is also one of the test phases.

During the test planning process, each proposed test phase is identified and communicated to the project team. This process helps the team to understand the overall testing approach.


Notes

Testing Phase Criteria

Two important concepts connected with the test phases are the entrance and exit criteria. Each phase must have a defined criterion that objectively and absolutely declares whether the phase is over and the next one has begun.

The tests to be carried out are also planned during the test phase. The tests to be planned involve:

1. ***Unit Tests:*** Deal with small units or modules of software.
2. ***Integration Tests:*** Deal with several units or modules that combine to form a subsystem.
3. ***System Tests:*** Deal with testing the entire software system.

It is obligatory upon planners to consider the following issues before beginning a specific test plan:

1. What to test?
2. Which sources should be used for test cases?
3. Who should perform the tests?
4. Where to perform the tests?
5. When to terminate the tests?

A straightforward approach to test software recommends developing a complete and comprehensive software test plan that requires performing unit tests for all the individual units, integration tests for all the unit integrations, and a system test to test the software system as a whole. By adopting this approach, one can ensure top quality software. However, this requires the investment of vast resources and an extended timetable.

10.4 Strategy

The test strategy is a document that describes the approach that the test team adopts to test the software, both overall and for each phase. Assume that you have been given a code that needs to be tested. There may be times when it might be better to test some part of the code manually and the remaining part of the code with tools and automation. If you decide to use tools, then it should be ascertained whether the tools need to be developed or if existing commercial solutions can be purchased. It may also be more efficient to outsource the entire testing process to a specialized testing company and commission only a small testing team to supervise the work.

Deciding on the strategy is a complicated task. This needs to be decided by experienced testers because it can determine the success or failure of the testing process. It is critical for everyone in the team to understand and be in agreement with the proposed plan. The test environment is finalized at this stage. The test environment includes the platforms on which testing should be performed.

10.5 Resource Requirements

Planning the resource requirements is the activity of deciding the requirements to accomplish the testing strategy. Everything that could possibly be needed throughout the testing process needs to be considered here. Some of the resource requirements that must be considered during the testing process are given below:

1. ***People:*** Number, experience, expertise; should they be full-time, part-time, contract, and so on.
2. ***Equipment:*** Computers, test hardware, printers, tools, and so on.
3. ***Office and Lab Space:*** Location, size/area, arrangement.
4. ***Software:*** Word processors, databases, custom tools. What needs to be purchased and what needs to be written.
5. ***Outsource Companies:*** Will they be employed? What will be the basis for choosing them? How much will they cost?

6. ***Miscellaneous Supplies:*** Disks, phones, reference books, training material. And other things that might be necessary during the course of the project.

The resource requirements are specific and depend on the project, team, and company. The test plan needs to determine all the resources needed to test the software. Many a times it becomes difficult or even impossible to obtain resources during the later part of the project. Hence, it is important to consider and determine all the requirements when creating the list of resources.

10.6 Testing Schedule

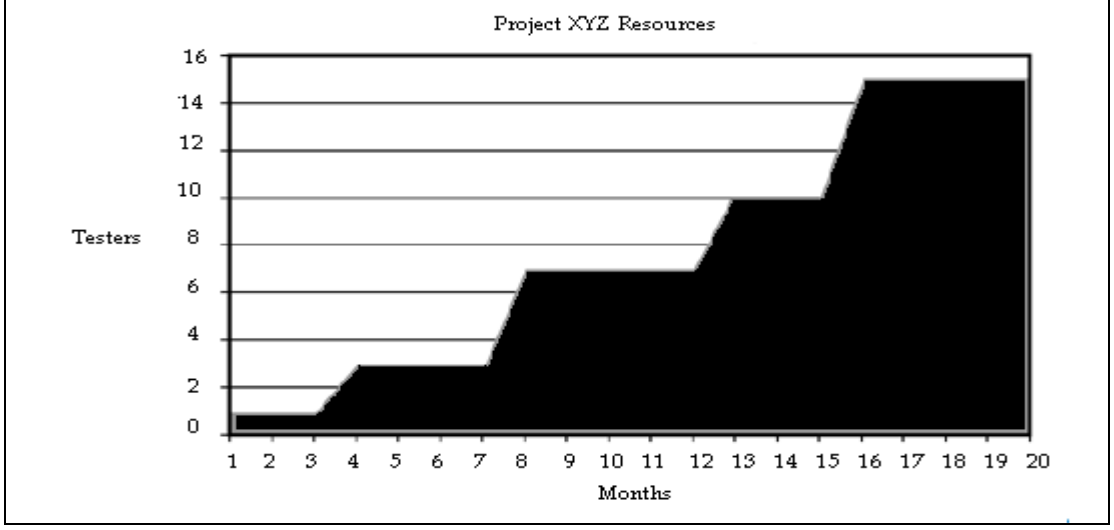
Testing schedule is a document wherein all the execution dates, which are mapped to the overall project schedule, are presented. It is very important to plan a schedule for all tests to know the likely start and end dates for the testing process. Preparing test schedule also identifies those modules that were considered easy to code and design, but may be very time consuming to test.

Example: Assume that there are many menu options in an application. Many end users may not frequently use some menu options. No one may appreciate the testing effort that is put in testing such rarely used menu options This is because the testing time for these menu options in the application could be many weeks.

Thus, creating a test schedule, as part of test planning, will present the project manager with the information needed to schedule the testing appropriately. Based on the testing schedule, this information can also be used to decide the removal of certain menu options from the product or postponing their testing to a later release, or adding more testers to complete the testing faster.

The extent of testing should not be distributed uniformly over the entire product development cycle. Some testing takes place early in the form of specification and code reviews, tool development, and so on. The number of testing tasks, the people involved, and the amount of time spent in testing increases during the course of the project, with the peak being just before the product is released. Figure 10.2 shows what a typical test resource graph may look like.

Figure 10.2 A Typical Test Resource Graph



This gradual increase in the test effort indicates that the test schedule is influenced by prior happenings in the project. Suppose that some part of the project is delivered to the test team one week late and only two weeks were scheduled for testing, what happens then? Does this mean that the two weeks scheduled for testing will now have to take place in only one week or does the project get delayed by one week? This problem is known as schedule crunch.

One way of avoiding the testing tasks from being crunched is by avoiding absolute dates for starting and ending tasks in the test schedule.

Example: The table 10.1 shows an example of a test schedule that can be used by a testing team during test planning process.

Table 10.1 A Sample Test Schedule Based on Fixed Dates

Testing Task	Date
Test Plan Complete	4/7/2010
Test Cases Complete	7/3/2010
Test Pass #1	7/17/2010–9/2/2010
Test Pass #2	9/17/201011/2/2010
Test Pass #3	11/17/2010–12/16/2010

If relative dates that are defined by the entrance and exit criteria of the testing phases are used in the test schedule, it becomes clearer that the testing tasks rely on some other deliverables being completed first. The time taken by the individual tasks also becomes more noticeable.

Example: The table 10.2 shows an example of a test schedule with relative dates.

Table 10.2 A Sample Test Schedule Based on Relative Dates

Testing Task	Start Date	Duration
Test Plan Complete	9 days after specification complete	5 weeks
Test Cases Complete	Test plan complete	15 weeks
Test Pass #1	Code complete build	8 weeks
Test Pass #2	Beta build	8 weeks
Test Pass #3	Release build	4 weeks

There are many software scheduling products that will make this process easier to manage. The project manager or test manager is responsible for the schedule and is likely to use such software scheduling products to create test schedules.

10.7 Test Cases

The test case is the heart of a test plan. A test case is a document that describes a set of data inputs and operating conditions required to run a test, together with the expected results of the run. The tester should run the software according to the test case documentation and then compare the actual results with the expected results noted in the documents. If the obtained results are in complete agreement with the expected results, then it indicates that no error has been identified. The result of such a test case is said to be 'pass'. A potential error is said to be identified when some or all of the results do not agree with the expected results. In such cases, the test case will have the status 'fail'.

Table 10.3 Example of a Test Case

Test Case No.	Test Name	Description	Expected Results	Pass/Fail
1	C6455_SA112 on Linux. Tests the voice channel with the new out file version and the previous Sample Application version.	Run the previous voice fax Sample Application version with the new out file version. Run the voip1.txt script.	The script runs successfully. Good quality voice is heard in both legacy phones.	
2	C6455_SA112 on Linux. Tests the fax channel with the new out file version and the previous Sample Application version.	Run the previous voice fax Sample Application version with the new out file version. Run the foip1.txt script.	The script runs successfully. Fax pages are transmitted from both fax machines.	

Test cases will be discussed in detail in the subsequent sections.

10.8 Bug Reporting

Bug reporting is the process of describing the problems found during the execution of the test case. These could be done on paper, but using a database to track them is a lot easier and efficient. It is important to plan the process used to manage the bugs so that every bug is tracked from the point it is found to the point it is fixed. More will be discussed about bugs in subsequent sections.

10.9 Metrics and Statistics

Metrics and statistics are the means by which the development and success of the project and the testing activity are determined. The test planning activity should exactly determine what information will be gathered, what decisions will be taken, and who will be responsible for gathering them.

Test metrics help us analyze the current progress in testing and give directions on how to proceed with the testing activities. They also allow us to set goals and predict future developments, and act as a feedback mechanism to improve the testing process.

Example: Some practical examples of test metrics are:

1. Total number of bugs found daily over the course of the project.
2. Record of bugs that still need to be fixed.
3. Current bugs ranked based on their severity.
4. Total bugs found per tester.
5. Total number of bugs found per software feature or area.

Case Study

Trooper Ltd. deploys HP Quality Center and saves $400,000 annually

Trooper Ltd. is a leading pay-television solutions provider, which has offices in 12 countries. The company witnessed substantial growth in the past few years. Consequently, there were many quality testing tools and processes that existed across the organization, creating a heterogeneous test environment.

Trooper Ltd. needed a standardized, centralized global quality testing platform to create more efficient testing processes, to lower costs, to accommodate changing manufacturing processes, and to improve cooperation and communication between the research and development centers.

To achieve this goal, testers and developers assessed many tools to establish a standard testing platform. The testers and developers then decided to adopt HP Quality Center (a Test Management Tool) as the standardized platform for all test development and management worldwide. This solution combines all tests into reusable sets, which lower the turn-around time for new set-top boxes.

Following improvements were observed after using the HP Quality Center to manage the written test cases:

1. Single, centralized, and efficient worldwide testing capability increased productivity.

2. On-demand customer status reports delivered proof of testing.

3. Embedded systems and customized test automation tools could be easily tested saving time and money.

The company was able to save $400,000 annually because of this efficient testing practice. The quality of the product was improved to a greater extent and the customer expectations could be met almost immediately. The issues related to quality were able to be resolved due to improved communication and cooperation.

Questions

1. How did Quality Centre help Trooper Ltd.?

2. "By using Quality Center to manage the written test cases, many improvements were observed." Justify.

Adapted from http://h20195.www2.hp.com/v2/getdocument.aspx?docname=4AA3-2682EEW.pdf

10.10 Summary

- Test planning is the basic test documentation that contains the record of the testing effort.
- The goal of a test plan is to facilitate communication between the tester and programmer. They cannot work in isolation.
- The IEEE Standard 829 for Software Test Documentation states that the purpose of a software test plan is to prescribe the scope, approach, resources, and schedule of the testing activities.
- To plan the test stages, the test team analyzes the selected development model and decides whether certain stages of testing should be performed during the course of the project.
- Test strategy is a document that describes the steps and actions that the test team will take during the course of the project.
- The testing schedule outlines the duration of each test that will be performed on the module.

10.11 Keywords

Database: A database is a collection of data and a system intended to organize and retrieve huge amounts of data quickly and easily.

Performance Metrics: The purpose of performance metrics is to measure an organization's activities and performance.

Test Cases: A test case is a detailed step by step instruction that seeks information about an aspect or a feature.

Code-and-Fix Model: The code and fix model is a way of software model which originated from the big-bang model.

10.12 Self Assessment

1. State whether the following statements are true or false:
   (a) There can be several test phases in the waterfall and spiral models.
   (b) Test strategy cannot determine the success or failure of the testing process.
   (c) Software Test Documentation defined by IEEE 829 states that the purpose of a software test plan is: "To prescribe the scope, approach, resources, and schedule of the testing activities".
2. Fill in the blanks:
   (a) The objective of implementing testing is to assess the effectiveness and _____ of the software.
   (b) A straightforward approach to test software recommends developing a comprehensive _____.
   (c) A detailed _____ will allow a tester to understand exactly what will be tested and how it will be tested.
3. Select the suitable choice for every question:
   (a) Testing _____ is a document wherein all the execution dates are presented which is mapped to the overall project schedule.
      (i) Plan
      (ii) Procedure
      (iii) Schedule
      (iv) Design specification

(b) A test design should have a list of:

(i) Identifiers and Features to be tested

(ii) Identifiers and Equipments

(iii) Tracking and Test proofs

(iv) Features to be tested

(c) The first stage of Test plan is:

(i) Determining the test methodology

(ii) Designing the test

(iii) Planning the test schedule

(iv) Planning the test cases

10.13 Review Questions

1. Assume that you have been assigned to test a banking application of user login authentication. Elaborate the steps that you would follow. Use the steps of Testing as a guideline
2. "Planning the resource requirements is the activity of deciding the requirements to accomplish the testing strategy." What are the aspects of resource requirements that you would consider?
3. "The test case is the heart of a test plan." Explain the significance of test cases
4. Assume that you have been assigned the task of measuring the effectiveness of testing. What methods and technique would you suggest?

Answers: Self Assessment

1. (a) True (b) False (c) True
2. (a) Efficiency (b) Planning (c) Test plan
3. (a) Schedule (b) Identifiers and Features to be tested
   (c) Determining the test methodology

10.14 Further Readings


Books

Daniel Galin, Software Quality Assurance-From theory to implementation, Pearson

Ron Patton, Software Testing, Second Edition, Sams Publishing


Online link

http://www.exforsys.com/tutorials/testing/bug-life-cycle-guidelines.html

http://www.scribd.com/doc/16103218/Software-Testing-Manual

http://www.nickjenkins.net/prose/testingPrimer.pdf

Unit 11: Test Case Planning

CONTENTS

Objectives

After studying this unit, you will be able to:

- Explain the test case design procedure
- Analyze bug life cycle

Introduction

We are aware that test cases are pivotal to any Test Plan. They help us discover information about the product. According to IEEE Standard 610 Test cases are defined as "A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement." Another definition from IEEE Standard 829 states that test cases are "Documentation specifying inputs, predicted results, and a set of execution conditions for a test item."

Let us now learn the different definitions of test cases. According to Ron Patton, "Test cases are the specific inputs that you'll try and the procedures that you'll follow when you test the software." Boris Beizer defines a test as "A sequence of one or more subtests executed as a sequence because the outcome and/or final state of one subtest is the input and/or initial state of the next." The word 'test' is used to include subtests, tests proper, and test suites.

Perhaps, a simpler definition is stated by Brian Marick, who calls the test case as test idea, where he defines as follows, "A test idea is a brief statement of something that should be tested. For example, if you're testing a square root function, one idea for a test would be 'test a number less than zero'. The idea is to check if the code handles an error case."

11.1 Test Cases

Testing becomes easier when systematic development models are used with formal documentation such as product specifications and design specifications. The testing process proves to be efficient and predictable when disciplined development models are used.

Example: In a code-and-fix type of model, the testers have to often guess which testing to perform and whether what they find are indeed bugs.

If testers want the whole software development process to be disciplined, they must work towards developing some methods and rules that will help the process to run more smoothly. Precise and systematic planning of test cases is a step in that direction. Acting accordingly is important for four reasons:

1. ***Organization:*** It is possible to have thousands of test cases even for small projects. Several testers may have been involved in creating the test cases over the course of several months or even years. Good planning organizes the test cases so that all the testers in the team review and use them effectively.
2. ***Repeatability:*** It is necessary to run the same tests several times over the course of the project to find new bugs and ensure that old ones have been fixed. Improper planning makes it difficult to know which test cases were last run and exactly how they were run, so that the exact tests could be repeated.
3. ***Tracking:*** Over the course of the project, the tester should be able to answer the questions like:
   (a) How many test cases were planned to be run?
   (b) How many were run on the last software release?
   (c) How many passed and how many failed?
   (d) Were any test cases skipped?

   If no planning went into running the test cases, it would be impossible to answer these questions.
4. ***Proof of Testing (or Not Testing):*** In a few high-risk projects, the software test team must confirm that it did indeed run the tests that it planned to run. It could actually be illegal and hazardous to release software in which a few test cases were skipped. Suitable test case planning and tracking provides a means for proving what was tested.

Did you know? One type of software testing, known as 'ad hoc testing' describes performing tests without a real plan. Here, neither is a test case planned nor is a high-level test plan created. Some testers are naturally good and can find bugs immediately even without a test plan.

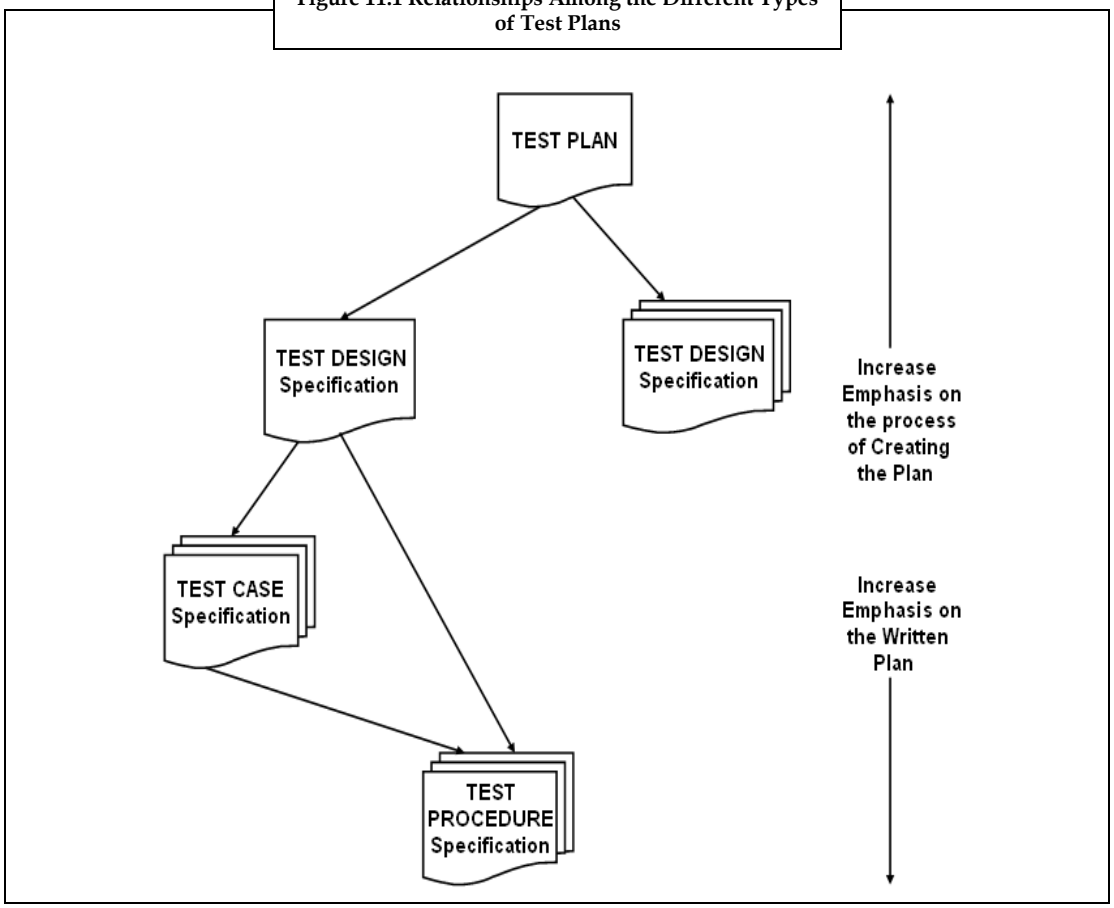
Task Note down a brief outline for the test plan you would use to test a property taxing system. Also, mention the expected results for each of the test cases.

11.1.1 Test Case Planning

By now you must be familiar with the project level test plan. The next three levels; namely: the test design specification, the test case specification, and the test procedure specification are discussed in detail in the subsequent sub-sections. The different levels of test documents interact and vary with respect to their importance on the document itself or the process of creating it.

Figure 11.1 shows the relationships among the different types of test plans.

Figure 11.1 Relationships Among the Different Types of Test Plans



As shown in Figure 11.1, tracing down from the top-level test plan, less emphasis is given to the process of creation and more to the resulting written document. The reason for this is that the test plans become so useful that they are used on a daily, sometimes hourly, basis by the testers performing the test.

The information on test planning provided in this chapter is adapted from the IEEE 829 Standard for Software Test Documentation (available on standards.ieee.org). This standard is widely used by many testing teams as test planning documentation, because it represents a logical and reasonable method for test planning. The important thing to understand about this standard is that it should be used as a guideline and not a standard. You are bound to follow it strictly only when the type of software you are testing demands it or if your company or industry policy instructs you to follow the standards.

However, what used to be presented as a written document can be better and more efficiently presented today as a spreadsheet or a database. The test team should create test plans that include the information outlined in IEEE 829. If written documents work best, then they are used. However, if a central database proves to be more efficient and there is time and budget to develop or buy one, this approach is adopted. This is because the four goals of test planning -- namely organization, repeatability, tracking, and proofing should always be met.


Caution

It is important that the test cases, whose expected result is an error, have conditions. Only by testing the software for such non-regular conditions can it be assured that the software does not produce undesirable and unexpected situations.

11.1.2 Test Design

The overall test plan for the project is written at a very high level. It breaks up the software into specific modules based on the features and testable elements and assigns them to individual testers, but it does not specify as to how those modules will be tested. There may be an indication of using automation, black-box, or white-box testing, but the test plan does not get into the details of exactly where and how they will be used.

IEEE 829 states that "the test design specification improves the test approach (defined in the test plan) and finds the modules to be covered by the design and its associated tests. It also identifies the test cases and test procedures, if any, required to accomplish the testing and specifies the feature pass/fail criteria."

The purpose of the test design specification is to organize and describe the testing that needs to be performed on a specific module. However, it does not provide the detailed cases or the steps needed to perform the testing. The following points, adapted from the IEEE 829 standard, address this purpose and should be part of the test design specifications that you create:

1. ***Identifiers:*** A unique identifier that can be used to refer and locate the test design specification. The specification should also refer to the overall test plan and should contain pointers to any other plans or specifications that it refers to.
2. ***Features to be Tested:*** These contain a description of the software feature covered by the test design specification. The example below illustrates the features to be tested for a calculator.

Example: These are some of the features of a calculator that need to be tested.

1. The addition function of Calculator
2. Font size selection and display in WordPad
3. Video card configuration testing of QuickTime

This section should also identify features that may be indirectly tested as a side effect of testing the primary feature.

Example: Although not the target of testing a calculator, the user interface of the file open dialog box will be indirectly tested in the process of testing the load and save functionality.

It should also list features that will not be tested and may be misunderstood as being covered by this plan.

Example: Testing the calculator's addition function will be performed with automation by sending keystrokes to the software. Hence, there will be no indirect testing of the onscreen user interface.

3. ***Approach:*** This includes a description of the general approach that will be used to test the features. It should give details about the approach, if any, listed in the test plan, describe the technique to be used, and explain how the results will be verified.

Example: A testing tool will be developed to sequentially load and save pre-built data files of various sizes. The number of data files, the sizes, and the data they contain will be determined through black-box techniques and supplemented with white-box examples from the programmer. A pass or fail will be determined by comparing the saved file bit-for-bit against the original, using a file compare tool.

4. ***Test Case Identification:*** These are high-level descriptions and references to the specific test cases that will be used to check the feature. It should list the selected equivalence partitions and provide references to the test cases and test procedures used to run them.

Example: Table 11.1 Shows a sample Test Case Identification

Table 11.1 A Sample Test Case Identification

Test Case Description	Identification Number
Check the highest possible value	Test Case ID# 17622
Check the lowest possible value	Test Case ID# 17623
Check several interim powers of 2	Test Case ID# 17624

It is important that the actual test case values are not defined in this section. For someone reviewing the test design specifications for proper test coverage, a description of the equivalence partitions is much more useful than the specific values themselves.

5. ***Pass/Fail Criteria***: This describes exactly what constitutes a pass and a fail of the tested feature. What is acceptable and what is not? The criteria followed is as given:

   (a) A pass is when all the test cases are run without finding a bug.

   (b) A case of failure of test condition happens when 10% or more of the test cases fail.

However, there must be a clear criteria based on which the pass or fail of a feature is determined.

11.1.3 Writing Test Cases

IEEE 829 states that the test case specification "documents the actual values used for input along with the anticipated outputs. A test case also identifies any constraints on the test procedure resulting from use of that specific test case."

Fundamentally, the details of a test case should explain exactly what values or conditions will be given to the software and what result is expected. It can be referenced by one or more test design specifications or may reference more than one test procedure. The IEEE 829 standard also lists some additional information that needs to be included.

1. ***Identifiers:*** A unique identifier is referenced by the test design specifications and the test procedure specifications.
2. ***Test Item:*** This describes the detailed feature, code module, and so on that's being tested. It should be more specific than the features listed in the test design specification.

Example: If the test design specification states: "the addition function of Calculator," then the test case specification would state that the upper limit overflow handling of addition calculations.

It should also provide references to product specifications or other design documents on which the test case was based.

3. ***Input Specification:*** This specification lists all the inputs or conditions given to the software to execute the test case.

Example: While testing a calculator, it may be as simple as 1+1. However, while testing a cellular telephone switching software, there could be hundreds or thousands of input conditions. While testing a file-based product, it would be the name of the file and a description of its contents.

4. ***Output Specification:*** This describes the result you expect from executing the test case. Did 1+1 equal 2? Were the thousands of output variables set correctly in the cell phone software? Did all the contents of the file load as expected?
5. ***Environmental Needs:*** Environmental needs are the hardware, software, test tools, facilities, staff, and so on that are necessary to run the test case.
6. ***Special Procedural Requirements:*** This section describes anything unusual that must be done to perform the test.

Example: Testing WordPad probably doesn't need any extraordinary procedures, whereas testing nuclear power plant software needs extraordinary procedural preparations.

7. ***Intercase Dependencies:*** If a test case depends on another test case or might be affected by another, the information is included here.

If this suggested level of documentation is strictly followed, at least a page of explanatory text for each test case you identify should be written. Thousands of test cases could result in thousands of pages of documentation.

This is another reason why the IEEE 829 standard should be taken as a guideline. Government projects and certain industries are required to document their test cases to this level. However, in other instances, you can adopt an alternative approach. Taking an alternative approach does not mean dismissing or neglecting important information—it means figuring out a way to abridge the information into a more efficient means of communication.

Test cases can be presented in the form of matrix or table. Each line of the matrix represents a specific test case and has its own identifier. All the other information that goes with a test case, namely: test item, input specification, output specification, environmental needs, special requirements, and dependencies are most likely common to all these cases and can be written and attached to the table. Someone reviewing the test cases could read this information and then review the table to check for its coverage.

Example: Figure 11.2 shows an example of test cases for printer compatibility table.

Figure 11.2 A Sample Test Case for Printer Compatibility Table

Test Case ID	Printer Mfg	Model	Mode	Options
WP0001	Canon	BJC-7000	B/W	Text
WP0002	Canon	BJC-7000	B/W	Super photo
WP0003	Canon	BJC-7000	B/W	Auto
WP0004	Canon	BJC-7000	B/W	Draft
WP0005	Canon	BJC-7000	Color	Text
WP0006	Canon	BJC-7000	Color	Super photo
WP0007	Canon	BJC-7000	Color	Auto
WP0008	Canon	BJC-7000	Color	Draft
WP0009	HP	LaserJet IV	High	
WP0010	HP	LaserJet IV	Medium	
WP0011	HP	LaserJet IV	Low	

Other ways of presenting test cases are simple lists, outlines, or even graphical diagrams such as state tables or data flow diagrams. The goal is to communicate your test cases to others and hence you should use whichever method is most helpful.

11.1.4 Test Procedures

IEEE 829 states that the test procedure specification identifies all the steps required to operate the system. It describes how the test will be run, the physical set-up required, and the procedure steps that need to be followed.

The test procedure or test script specification specifies the step-by-step details on how to perform the test cases. Here is the information that needs to be specified:

1. ***Identifier:*** A unique identifier that binds the test procedure to the related test cases and test design.
2. ***Purpose:*** The purpose of the procedure and indications to the test cases that it will execute.
3. ***Special Requirements:*** Other procedures, special testing skills, or special equipment considered necessary to run the procedure.
4. ***Procedure Steps:*** Thorough description of how the tests are to be run.
5. ***Log:*** Tells you how and by what method the results and observations will be recorded.
6. ***Setup:*** Gives details on how to prepare for the test.
7. ***Start:*** Gives details on the steps used to start the test.
8. ***Procedure:*** Explains the steps used to run the tests.
9. ***Measure:*** Explains how the results are to be determined.
10. ***Shut Down:*** Describes the steps for suspending the test for unexpected reasons.
11. ***Restart:*** Assists the tester in picking up the test from a certain point in case of failure, or after shutting down.
12. ***Stop:*** Explains the steps for a systematic halt to the test.

13. ***Wrap Up:*** Tells how to restore the environment to its pre-test condition.
14. ***Contingencies:*** Tells what to do if things do not go as planned.

It is not enough for a test procedure to direct a tester to try all the test cases and get back with the observations. The test procedure should instruct a new tester on the way to perform the testing. A detailed procedure helps a tester know what exactly needs to be tested and how.

Example: Figure 11.3 shows an excerpt from a fictional example of a test procedure for Windows Calculator.

Figure 11.3 Sample Test Procedure

Purpose: This procedure describes the steps necessary to execute the Addition function test cases WinCalc98.0051 through WinCalc98.0185.
Special requirements: No special hardware or software is required to run this procedure other than what is outlined in the individual test cases.
Procedure Steps:

Log: The tester will use WordPad with the Testlog template as the means for taking notes while performing this procedure. All the fields marked as required must be filled in. A bug tracking system will be used to record any problems found while running the procedure.

Setup: The tester must install a clean copy of Windows 98 on his or her machine prior to running this procedure. Use the test tools WipeDisk and Clone before installing the latest version of Windows 98.

Start:

Boot up Windows 98.
Click the Start Button.
Select programs.
Select Accessories.
Select Calculator.

Procedure: For each test case identified above, enter the test input data using the keyboard (not the onscreen numbers) and compare the results to the specified output.

Source: Ron Patton, Software Testing, Second Edition, Sams Publishing

11.1.5 Test Organization and Tracking

An important consideration that needs to be taken into account when creating the test case documentation is the organization and tracking of information. Some questions that a tester or the test team should be able to answer may include:

1. Which are the test cases you plan to run?
2. What is the number of test cases you plan to run? What will be the time taken to run them?
3. Can you decide on choosing the test suites (groups of related test cases) to run particular features of the software?
4. When the test cases are run, will you be able to record pass and fail result of each test case?
5. Of the cases that failed, which ones failed the last time too?
6. What is the percentage of the test cases that passed?

These types of questions might be asked over the course of a typical project. However, initially some sort of a process needs to be in place that allows you to manage your test cases and track the results of running them. There are basically four types of systems. They are:

1. ***In Your Head:*** You must not consider this, even for the simplest projects, unless you are testing software for your own personal use and you do not want to track your testing.
2. ***Paper/Documents:*** It is possible to develop test cases on paper for very minor projects. Tables and charts of checklists have proved to be effective. The advantage of maintaining a written document is that if a written checklist includes a tester's initials or signature indicating that the tests were run, it serves as an excellent proof in the court-of-law that testing was performed.
3. ***Spreadsheet:*** A preferred and practical method of tracking test cases is by using a spreadsheet. A spreadsheet can provide a quick view of the status of your testing. This is because all details concerning the test cases are maintained in one place. Spreadsheets are easy to set up and use. They behave as a good proof of testing. A spreadsheet can be used to effectively track and manage test suites and test cases.

Example: Figure 11.4 shows an example of a spreadsheet application used to manage test suites and test cases.

Figure 11.4 A Sample Spreadsheet

Microsoft Excel - Purple Dinosaur Test Tracking.xls

File Edit View Insert Format Tools Data Window Help

	A	B	C	D	E
1		Purple Dinosaur Test Tracking			
2	Test Suite /Cases	Test Pass 10/15/1997	Test Pass 11/30/1997	Test Pass 1/5/1998	Bug ID List
3	**Basic Hardware Functionality**				
4	Left Arm Motion	Pass	Pass	Pass	
5	Right Arm Motion	Pass	Pass	Pass	
6	Head Motion	Fail	Pass	Pass	12
7	Touch Sensors	Pass	Pass	Pass	
8	Peek-a-Boo Sensor	Pass	Pass	Pass	
9	PC Radio Transmission	Fail	Fail	Pass	19, 22
10	PC Radio Reception	Pass	Pass	Pass	
11	TV Radio Transmission	Pass	Pass	Pass	
12	TV Radio Reception	Pass	Pass	Pass	
13	**Summary**	**FAIL**	**FAIL**	**PASS**	
14					
15	**Basic Software Functionality**				
16	Songs	Pass	Pass	Pass	
17	Games	Fail	Pass	Pass	13
18	Peek-a-Boo	Pass	Pass	Pass	
19	Cleanup Song	Pass	Pass	Pass	
20	Timeout Sleep	Pass	Pass	Pass	
21	Commanded Sleep	Pass	Pass	Pass	
22	VCR Broadcast Mode	Pass	Fail	Fail	14, 29
23	PC Single Unit Mode	Pass	Pass	Pass	
24	**Summary**	**FAIL**	**FAIL**	**FAIL**	
25					

Test Case Summary

Source: Ron Patton, Software Testing, Second Edition, Sams Publishing

4. ***Custom Database:*** The best way to track test cases is to use a Test Case Management Tool. It is a database, programmed specifically to handle test cases. There are many commercially available applications that can be set up to perform just this task. If you want to create your own tracking system, then database software like FileMaker Pro, Microsoft Access, and many others provide almost drag-and-drop database creation facility. These software allow you to build a database that is mapped to the IEEE 829 standard in a very short time.

Few examples of test case management tools are:

Example:

1. *HP Quality Centre:* It is a web based test management software application.
2. *qaManager:* It is a simple web based application for managing QA Projects/Teams effectively and efficiently.
3. *ApTest:* It improves the consistency and control throughout the testing process.
4. *InformUp:* A simple application life cycle management tool.
5. *Testopia:* It is a generic tool for tracking test cases.

An important thing to know is that the number of test cases is usually in thousands and without a means to manage them, you and the other testers could find yourselves lost in large volumes of documentation.

11.2 Bug's Life Cycle

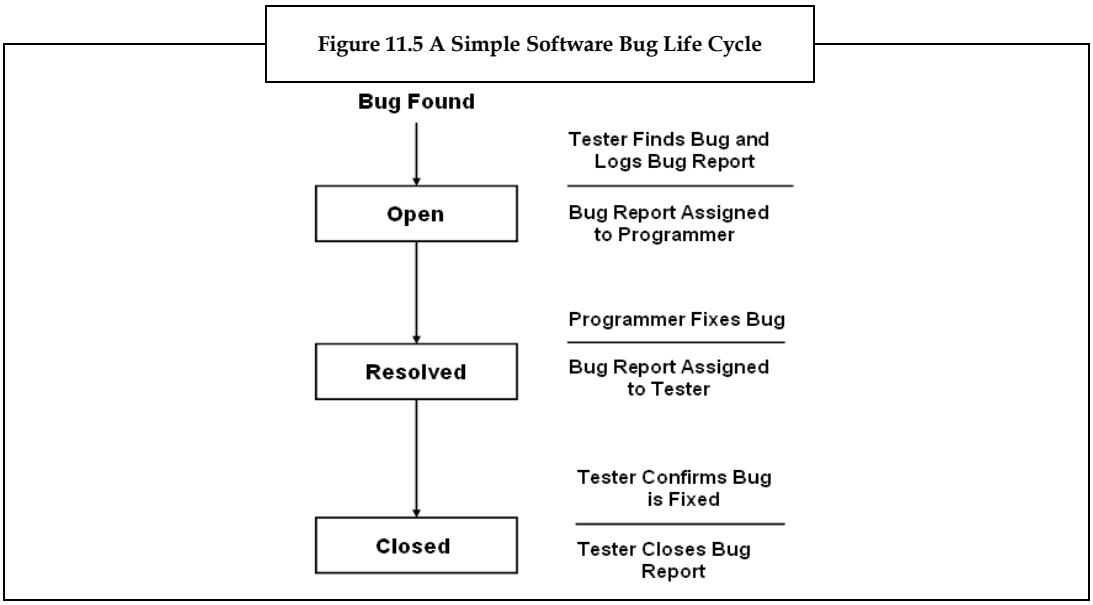
In software, a bug is a coding error or mistake. These bugs must be removed or fixed before the software is put to use. The activity of finding bugs in a program begins after the coding process is completed. It is then integrated with other units of programming to form a software product. The concept behind software bug life stages is similar to a real bug's life stages.

11.2.1 Stages of a Bug

In the software development process, a bug is considered to have a life cycle. In order to be fixed or closed, the bug should complete the life cycle. A specific life cycle guarantees that the process is standardized. The bug reaches different forms during the life cycle.

Example: Figure 11.5 shows an illustration of a simple software bug life cycle.

Figure 11.5 A Simple Software Bug Life Cycle

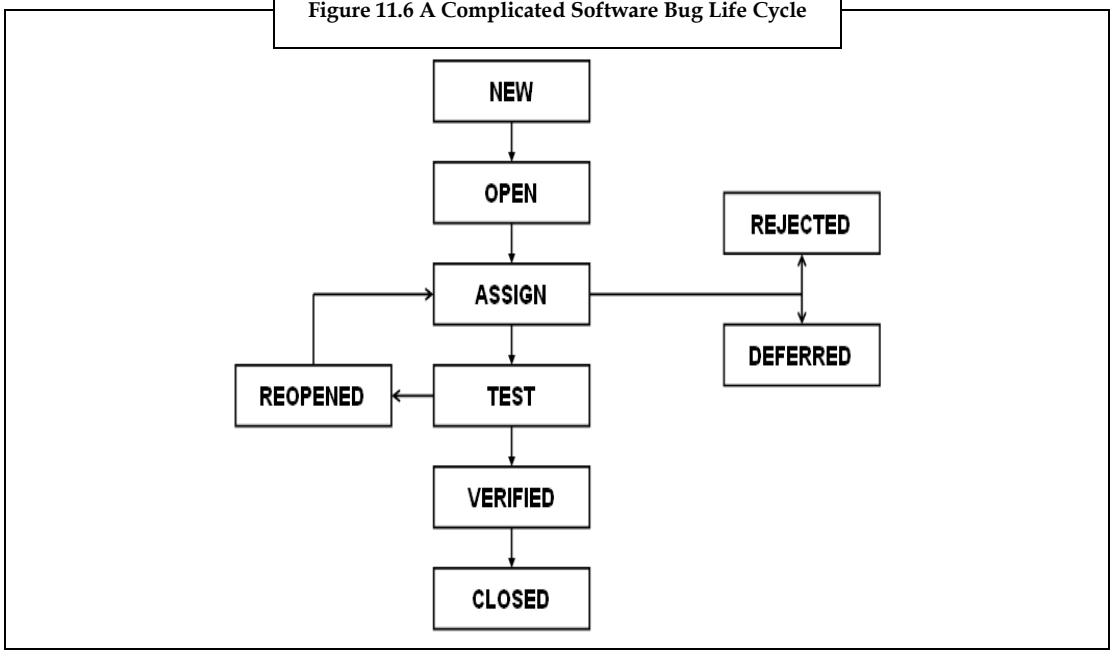


The example illustrates that when a bug is first found by a software tester, a report is recorded and given to a programmer to be fixed. This state is called the open state. After the programmer fixes the bug, he/she gives the report back to the tester and the bug now enters the resolved state. The tester then

carries out a verification test to confirm that the bug is indeed fixed. The bug then enters the closed state which is its final state.

In some cases, the life cycle gets a bit more complicated as illustrated in Figure 11.6.

Figure 11.6 A Complicated Software Bug Life Cycle



Source: http://www.exforsys.com/tutorials/testing/bug-life-cycle-guidelines.html

The different stages of a bug can now be summarized as follows:

1. ***New:*** A bug's state will be "NEW" when it is posted for the first time, that is, the bug is not yet accepted.
2. ***Open:*** After the tester posts the bug, the project manager agrees that the bug is legitimate and changes the state to "OPEN".
3. ***Assign:*** Once the project manager changes the state to "OPEN", the bug is given to the developer. Now the state of the bug is changed to "ASSIGN".
4. ***Test:*** Once the developer fixes the bug, the bug is given back to the testing team for the next round of testing. Before delivering the software with the bug fixed, the developer changes the state of the bug to "TEST". It indicates that the bug has been fixed and is given to the testing team.
5. ***Deferred:*** If the bug state says "DEFERRED", it means the bug is likely to be fixed in later releases. The rationale behind changing the bug to this state is influenced by many factors. Some factors maybe: the priority of the bug may be low, the software release date might be too close, and the bug may not have major effect on the software and so on.
6. ***Rejected:*** If the developer considers that the bug is not legitimate, the bug is rejected. Then the state of the bug is changed to "REJECTED".
7. ***Duplicate:*** When the bug is repeated two times or the two bugs refer to the same feature, then the status of one bug is changed to "DUPLICATE".
8. ***Verified:*** After the bug is fixed the status is changed to "TEST", and then the tester tests the bug. If the bug is not present in the software, then the tester attests that the bug is fixed and changes the status to "VERIFIED".
9. ***Reopened:*** If the bug is still found to exist even after the developer fixes the bug, the tester changes the status to "REOPENED". The bug goes through the complete life cycle once again.

10. ***Closed:*** Once the bug is fixed, it is again tested by the tester. If the tester believes that the bug is no longer present, the status of the bug is changed to "CLOSED". This state means that the bug is fixed, tested, verified, and validated.


Notes

Preventing defects proves to be much more effective and efficient in reducing the number of defects, but still most companies conduct defect discovery and removal. Discovering and removing defects is an expensive and incompetent process. From a company's point of view, it is effective to conduct activities that prevent defects.

11.2.2 Bug-Tracking System

The bug-reporting process is a complicated activity that requires a lot of information, a high level of detail, and a reasonable amount of discipline to be effective. Some type of system is needed to record the bugs found and to monitor them throughout their life cycle. A bug-tracking system maintains the record of the bugs found and helps to monitor the bugs throughout their life cycle.

The Standard: The Test Incident Report

The IEEE 829 Standard for Software Test Documentation defines a document called the Test Incident Report whose purpose is "to document any event that occurs during the testing process and which requires investigation", that is, "recording a bug." The following list, adapted and updated to reflect current terminology, shows the areas that the standard defines.

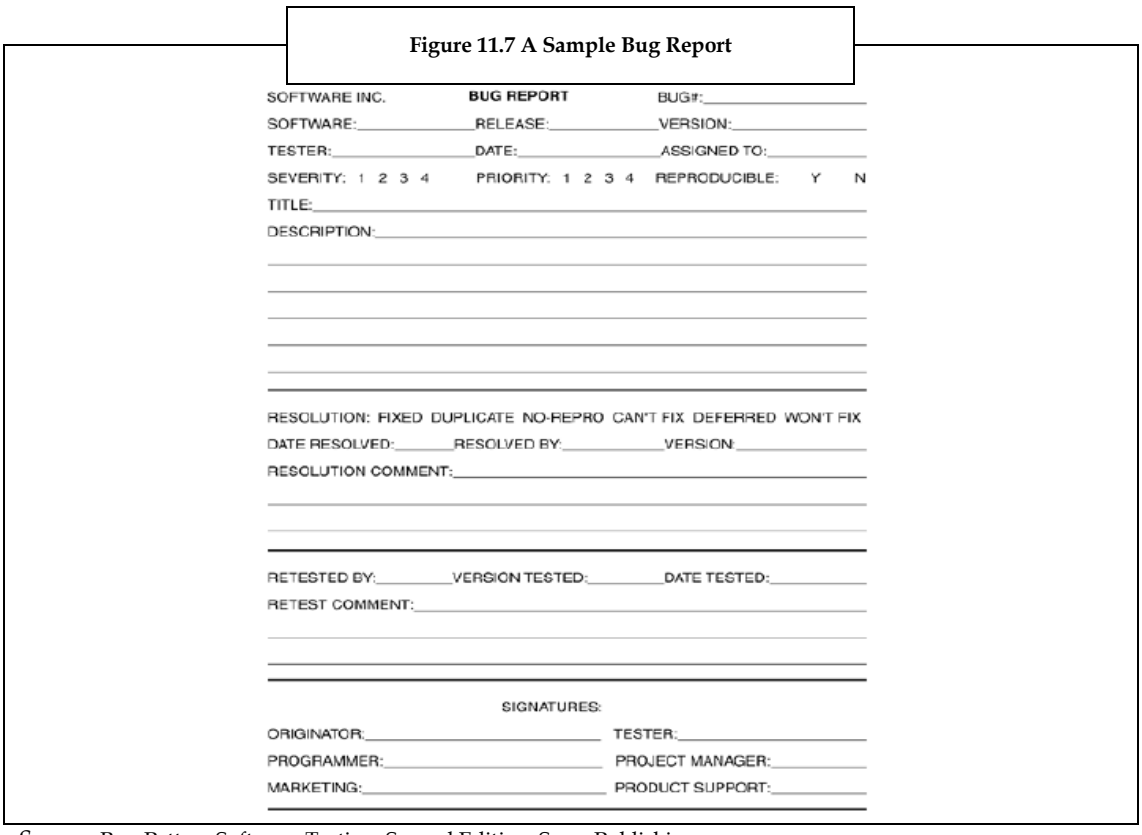
1. ***Identifier:*** Specifies a unique identification number which is used to locate the bug. This ID is exclusive to the bug report.
2. ***Summary:*** Summarizes the bug into a short, brief statement of fact. References to the software being tested and its version, the associated test procedure, test case, and the test specification should also be included.
3. ***Incident Description:*** Provides a detailed information of the bug with the following information:
   (a) Date and time
   (b) Tester's name
   (c) Hardware and software configuration used
   (d) Inputs
   (e) Procedure to be followed
   (f) Expected results
   (g) Actual results
   (h) Attempts to repeat and description of what was tried
   (i) Other observations or descriptions that may help the programmer find the bug
4. ***Impact:*** The severity and priority as well as an indication of the impact of the bug to the test plan, test specifications, test procedures, and test cases.

Manual Bug Reporting and Tracking

The IEEE 829 standard does not define the format that the bug report should follow, but it does give an example of a simple document.

Example: Figure 11.7 shows an example of how a paper bug report can look like.

Figure 11.7 A Sample Bug Report



SOFTWARE INC. **BUG REPORT** BUG#:_____

SOFTWARE:_____ RELEASE:_____ VERSION:_____

TESTER:_____ DATE:_____ ASSIGNED TO:_____

SEVERITY: 1 2 3 4 PRIORITY: 1 2 3 4 REPRODUCIBLE: Y N

TITLE:_____

DESCRIPTION:_____

RESOLUTION: FIXED DUPLICATE NO-REPRO CAN'T FIX DEFERRED WON'T FIX

DATE RESOLVED:_____ RESOLVED BY:_____ VERSION:_____

RESOLUTION COMMENT:_____

RETESTED BY:_____ VERSION TESTED:_____ DATE TESTED:_____

RETEST COMMENT:_____

SIGNATURES:

ORIGINATOR:_____ TESTER:_____

PROGRAMMER:_____ PROJECT MANAGER:_____

MARKETING:_____ PRODUCT SUPPORT:_____

Source: Ron Patton, Software Testing, Second Edition, Sams Publishing

Figure 11.7 shows how the details of a bug can be condensed to a single page of data. Observe that this single-page format can contain all the information required to identify and describe a bug. It also includes fields that can be used to keep track of a bug through its life cycle. Once all the details are filed by the tester, it can be given to a programmer for fixing the bugs.. There are fields provided in the form where the programmer can enter information regarding the fix. There is also an area wherein, after resolving the bug, the tester can provide information related to his efforts in retesting and closing out the bug. At the bottom of the form is an area for signatures. Here, the tester's name is written to indicate that a bug has been satisfactorily resolved.

Paper forms can be used without any problem for both small and mission-critical projects. But, the problem with paper forms is that it is inefficient for large scale testing.

Example: If someone wants to know the status of Bug #5529 or how many Priority 1 bugs were left to fix, then all the forms need to be checked manually in order to find the form containing the relevant information.

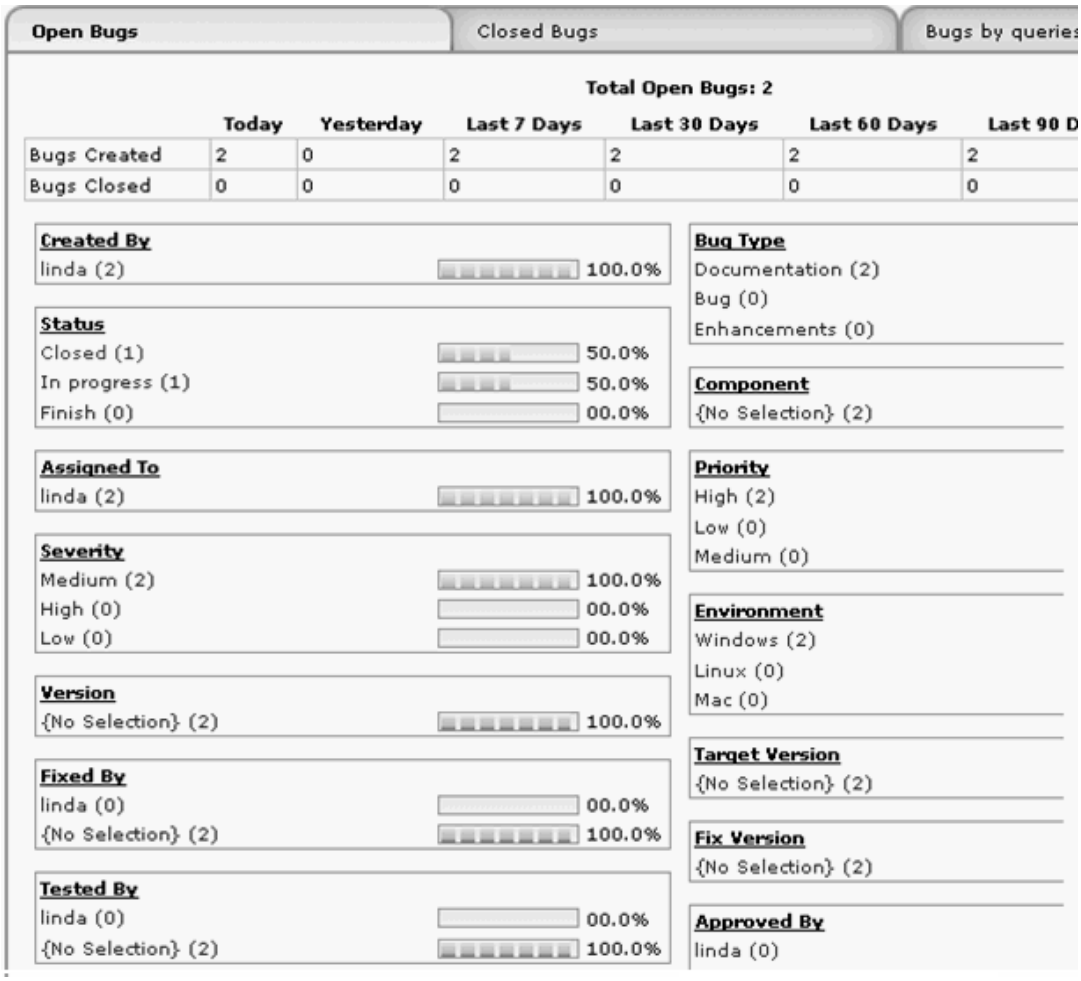
To avoid such a cumbersome activity, spreadsheets and databases can be used to record the status of bugs.

Automated Bug Reporting and Tracking

The IEEE 829 standard can serve as a guideline while using automated systems for bug reporting and tracking. As seen in figure 11.9, the information related to bug tracking is filled in the form which is primarily text and numbers. This information can easily be recorded on a database. The main window of a typical bug-reporting database shows what an automated system can provide.

Example: Figure 11.8 an automated bug reporting and tracking system.

Figure 11.8 A Sample Bug Reporting Database

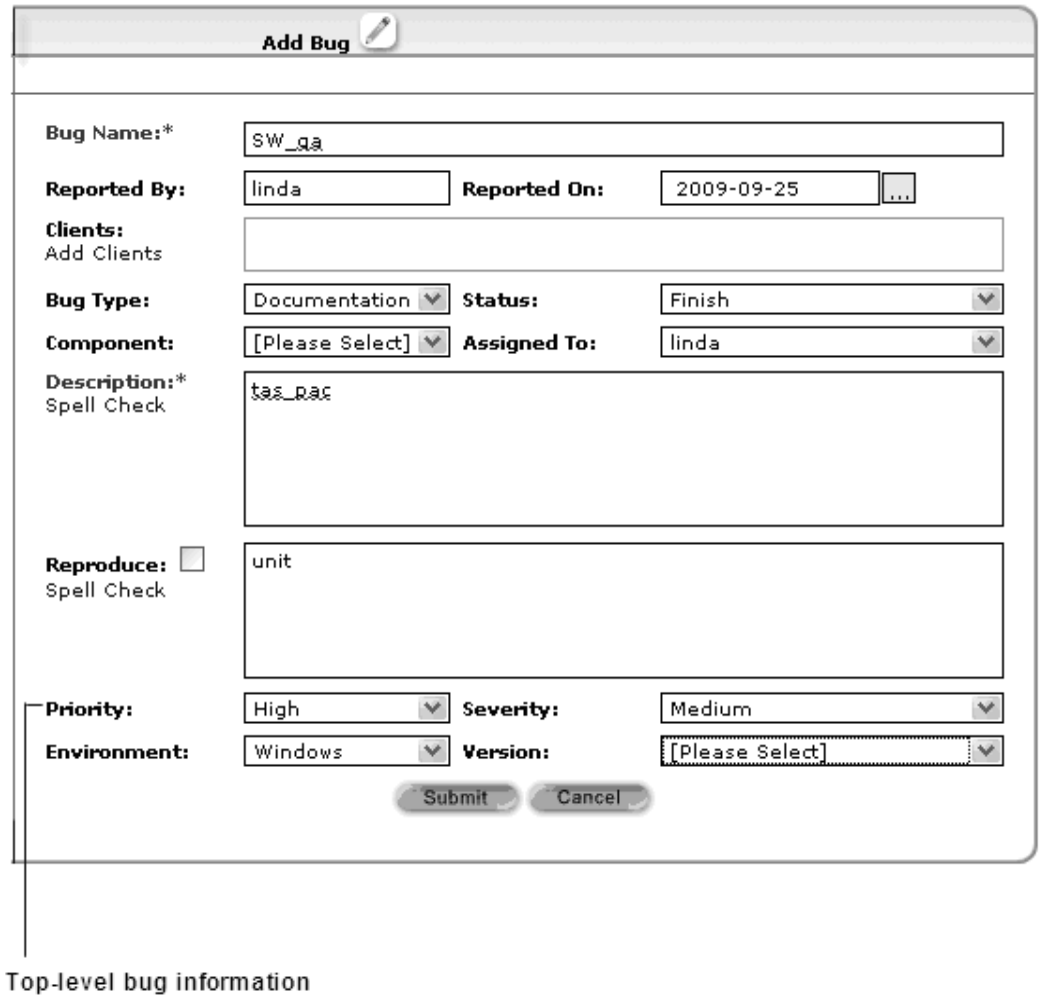


Source: http://www.bug-track.com/dashboard/viewDashboard.do?action=viewOpenBugsDashboard

Figure 11.8 illustrates a bug reporting and tracking database. A glance at the screen is enough to know the status of bugs. The bug's description includes its type, component, priority, environment, target version, fix version, and so on.

Example: Figure 11.9 shows the New Bug Dialog Box, in which information is entered to record a new bug into the system.

Figure 11.9 A Sample New Bug Dialog Box

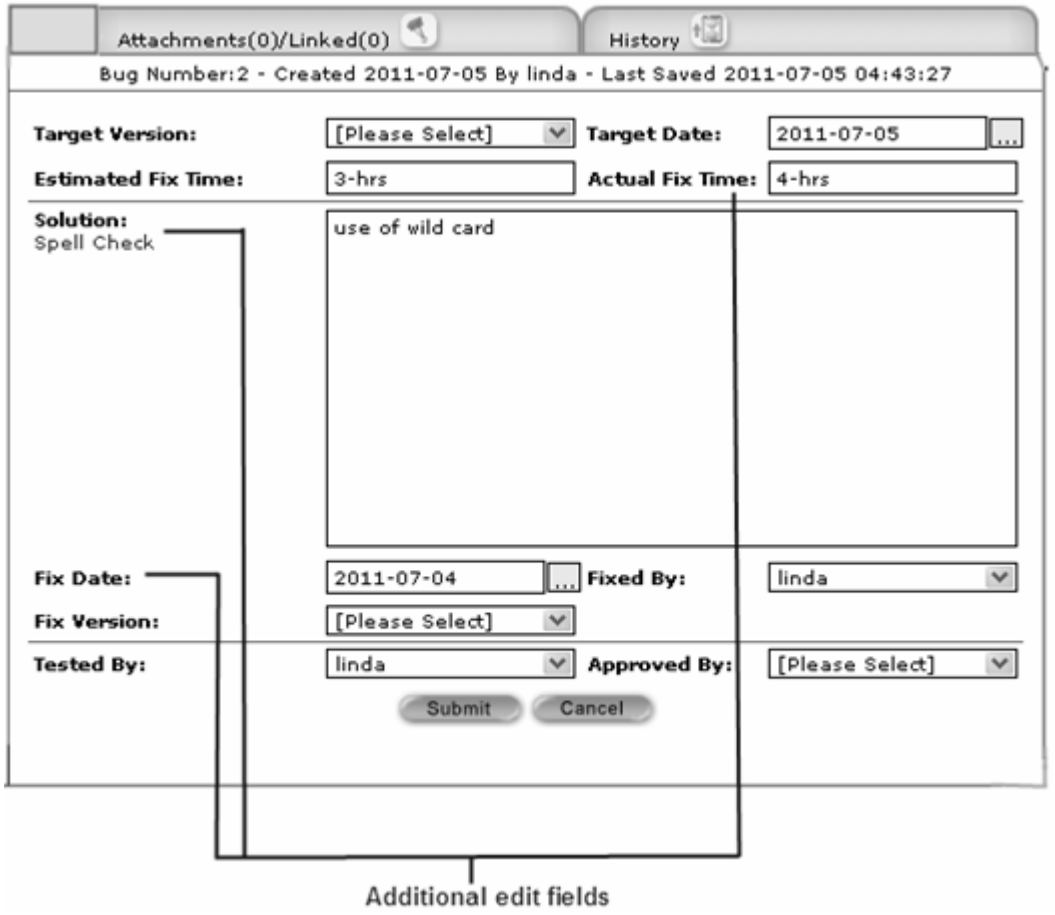


Source: http://www.bug-track.com/bugs/editBugs.do?action=Create

This dialog box is typically used by the programmer to record information concerning the bug fix. A drop-down list supplies different types of bug and their status. The bug is then given back to the tester for closing.

Example: Figure 11.10 shows the dialog box used when a bug is resolved by a programmer or project manager.

Figure 11.10 A Sample Dialog Box used when a Bug is Resolved by a Programmer or Project Manager



Source: http://www.bug-track.com/bugs/editBugs.do?action=Edit&key=148950&drill=&drill_key=null&Section=resolution&user_query_key=main&FromNavURL=&FromNav=&itemsOffSet=&scrollLoc=0#

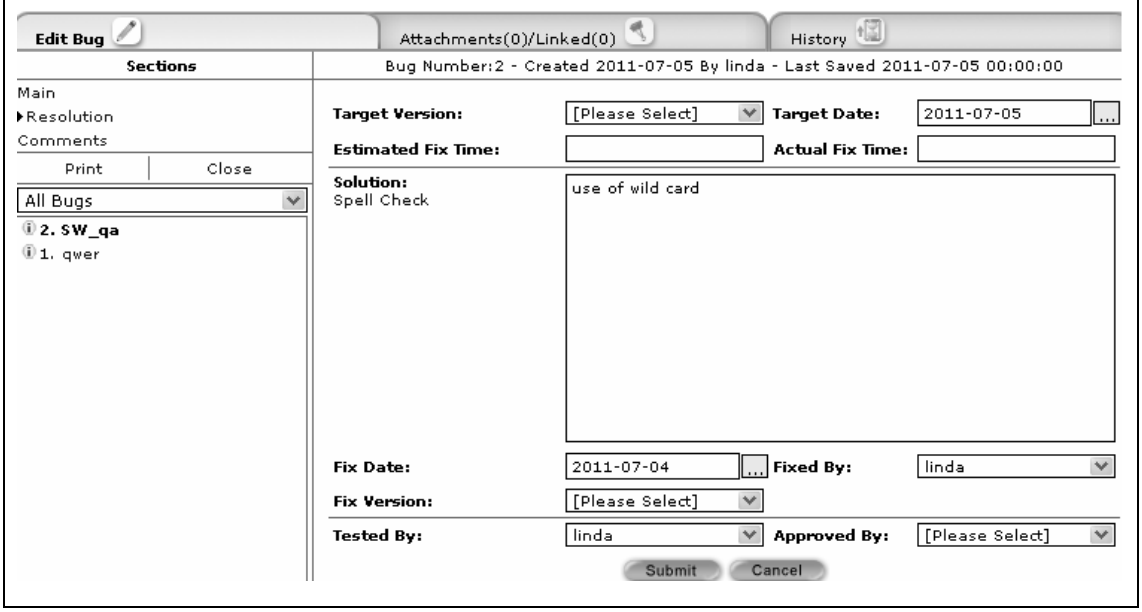
Several bug reporting and tracking databases track not just comments about the fix, but also details of what exactly the programmers did to make the fix. Herein, the line number, the module, and even the type of error can be recorded.

After a bug is resolved, it is given back to the tester for closing. Hence one will have a bug report containing the entire test history for review.

As the database has tracked every alteration to the bug report since it was opened, it helps to view the decisions that were made along the way and review what was fixed. It is possible that the bug was not fixed as expected, possibly a similar bug had been found and added by another tester, or maybe the programmer made a comment about the fix being risky. All this information will assist when a retest of the bug is done to make sure it is fixed. If it is found that it is not fixed, the bug is reopened to start the life cycle all over again.

Example: Figure 11.11 shows the bug closing dialog box.

Figure 11.11 A sample Bug Report, Ready for closing



Source: Mantis Bug Database Images Courtesy of Dave Ball and HBS International, Inc.

A bug-tracking database helps to organize the entire project team. By using a bug-tracking database, testers can communicate the status of the project, know who is assigned what task, and most importantly, assure that no bug slips occur.

Example: Microsoft Access and Bugzilla are very popular bug tracker databases. FogBugz is another example.

11.3 Summary

- Test planning is the basic test documentation that contains the record of the testing effort.
- The goal of a test plan is to facilitate communication between the tester and programmer. Both cannot work in isolation.
- The IEEE Standard 829 for Software Test Documentation states that the purpose of a software test plan is to prescribe the scope, approach, resources, and schedule of the testing activities.
- To plan the test stages, the test team analyzes the selected development model and decides whether certain stages of testing should be performed during the course of the project.
- The testing schedule outlines the duration of each test that will be performed on the module.
- The four features that make it necessary to plan the test cases are its organization, repeatability, tracking, and proof of testing.
- Test design specifications are used to organize and describe the testing that needs to be performed on a specific module.
- The main aim of a bug report is to let the programmer know that some part of the software is not giving the expected result.
- Bug tracking and reporting can be carried out manually or by using automated tools.

11.4 Keywords

Database: A database is a collection of data and a system intended to organize and retrieve huge amounts of data quickly and easily.

Log: It is a detailed record of events and actions.

Spreadsheet: Spreadsheets, also known as worksheets, have rows and columns that make it easy to display information to insert formulas and work with the data.

Testopia: Testopia is a test case management extension for Bugzilla, which is designed to be a generic tool for tracking test cases. It allows testing organizations to integrate bug reporting with their test case run results.

IEEE 829: IEEE 829-1998, also known as the 829 Standard for Software Test Documentation, is an IEEE standard. This specifies the form of a set of documents for use in eight defined stages of software testing. Each stage produces its own separate type of document.

11.5 Self Assessment

1. State whether the following statements are true or false:
   (a) IEEE 829 states that the test procedure specification identifies all the steps required to operate the system. It describes how the tester will physically run the test, the physical set-up required, and the procedure steps that need to be followed.
   (b) A bug retains the same form throughout its life cycle.
   (c) If the tester thinks the bug is legitimate, then the state of the bug is changed to "REJECTED".
   (d) The IEEE 829 standard does not define the format that the bug report should follow.
2. Fill in the blanks:
   (a) A detailed _____ will allow a tester to understand exactly what will be tested and how it will be tested.
   (b) A bug's state will be _____ when it is posted for the first time.
   (c) When the bug is accepted by the project manager, its state is _____.
3. Select the suitable choice for every question:
   (a) Planning test cases systematically is important for _____many reasons.
      (i) Two
      (ii) Three
      (iii) Four
      (iv) Five
   (b) IEEE 829-1998 Standard for Software Test Documentation is used widely by many testing teams because:
      (i) It is logical and reasonable
      (ii) It is easy to follow
      (iii) It is incorporated in a tool
      (iv) None of the above

(c) The test procedures document should specify a list of

(i) Procedures and Environmental needs

(ii) Special requirements and Procedure

(iii) Identifiers and Input specifications

(iv) Test items and Approach

(d) The state of the bug is set to "Deferred" when:

(i) The developer fixes the bug

(ii) The bug is still not fixed

(iii) The bug is repeated twice

(iv) The bug is decided to be fixed in the next release

11.6 Review Questions

1. "Precise and systematic planning of test cases is a step in making the testing process disciplined." Discuss the reasons for test case planning.
2. "IEEE 829 states that the test design specification improves the test approach (defined in the test plan) and finds the modules to be covered by the design and its associated tests." Elaborate.
3. "A bug-tracking database organizes the entire project team." Explain briefly how a bug tracking system helps.
4. IEEE 829 states that "the test procedure specification identifies all the steps required to operate the system. It describes how the tester will physically run the test, the physical set-up required, and the procedure steps that need to be followed." Explain.
5. "It is not enough for a test procedure to ask a tester to try all the test cases and report the observations. " Justify.
6. Do you think all bugs can be fixed? If not, list some reasons why some bugs might not be fixed.
7. "A preferred and practical method of tracking test cases is by using a spreadsheet." Discuss.
8. "One consideration that must be taken into account when creating the test case documentation is how the information will be organized and tracked." Do you agree? Justify.
9. "IEEE 829 standard is widely used by many testing teams" Explain why?
10. "If testers want the software development process to be disciplined, they use the four methods". Discuss.
11. "In some cases, the life cycle of a bug gets a bit more complicated." Do you agree? Justify.
12. "The IEEE 829 Standard for Software Test Documentation defines a document called the Test Incident Report." Briefly discuss the purpose of the Test Incident Report.

Answers: Self Assessment

1. (a) True (b) False (c) False (d) True
2. (a) Planning (b) New (c) Open
3. (a) Four (b) It is logical and reasonable

(c) Special requirements and Procedures (d) The bug is decided to be fixed in the next release

11.7 Further Readings


Books

Daniel Galin, Software Quality Assurance-From theory to implementation, Pearson

Ron Patton, Software Testing, Second Edition, Sams Publishing


Online link

http://www.exforsys.com/tutorials/testing/bug-life-cycle-guidelines.html

http://www.scribd.com/doc/16103218/Software-Testing-Manual

http://www.nickjenkins.net/prose/testingPrimer.pdf

Unit 12: Software Quality Assurance

CONTENTS

Objectives

After studying this unit, you will be able to:

- Define quality
- Explain testing and quality assurance at workplace
- Explain quality management in IT

Introduction

Quality is defined as the features or the attributes of the products that are appreciated by the end-users or the customers. "Quality means conformance to requirements "as indicated by Crosby in 1979.

Quality assurance is an orderly procedure of inspecting a particular product or a service that is being developed to meet the required standards. Many organizations allocate a whole unit for quality assurance purposes. A good quality assurance system not just enhances the organization's credibility, it also builds customer's belief, thereby improving the process which helps the organization to compete with others.

Did you know? During World War II, military weapons were checked and tested for defects after they were developed. However, in today's scenario quality assurance systems emphasize on identifying the defects before the development of the final product.

Quality assurance is considered to be the most important activity for any business involved in software development. The history of quality assurance in software development is similar to that of the history of quality in hardware manufacturing. Software quality assurance is defined as a planned and methodical pattern of actions used to ensure the quality of the product as per the standards established.

In order to follow quality guidelines, a company's management team frames quality assurance policies and objectives. The company's external consultant or management writes down the company policies and requirements in a structured format, as to how the staff can implement the quality assurance system. Once this guideline is framed and quality assurance procedures are implemented, an external evaluator examines the company's quality assurance system to ensure its conformance with the set standards such as ISO or CMM.

Example: In the year 2009 – 2010, Toyota Motors had recalled millions of its vehicles. This is because most of the vehicles had experienced an increased acceleration. This was due to incorrect placement of the driver's front floor mat at the foot pedal well. The wrong placement resulted in pedal entrapment when the vehicle was in motion. However, on investigation, it was found that the quality of the foot mat was low and this had caused the unintended acceleration in the vehicle.

12.1 Definition of Quality

"Quality is the enduring process of building and sustaining relationships by assessing, anticipating, and fulfilling stated and implied needs" Winder, Richard E. and Judd, Daniel K., 1996.

Quality can be defined as a state of being free from defects and deficiencies. It is achieved by adopting strict and consistent adherence to measure and verify the set standards to attain uniformity in the output which satisfies specific user requirements.

ISO defines quality as "the totality of features and characteristics of a product or service that bears its ability to satisfy stated or implied needs". It is the state of being free from defects and deficiencies by adopting strict and consistent adherence to measure and verify the set standards in order to attain uniformity in the output. This in turn satisfies specific user requirements.

To understand the meaning and importance of software quality in the software industry, we first need to understand the activities related to quality such as software quality assurance and software quality control.

Software Quality

When a software system or a process achieves a particular standard that satisfies the customer expectations, then the system is supposed to have achieved 'quality software' standard.

While measuring the features of a software product, two factors, namely: the quality of design and the quality of conformance are taken into consideration. Quality of design comprises a class of materials, the tolerance capacity, and the work specifications of the item. The quality of design is said to be fantastic when good class of materials are used to make the product. These materials are set to tolerate rigid circumstances and to perform at tough levels. The quality of a product can be judged by looking at the specifications.

When a product is set to execute its performance at high levels but performs really low, then it is understood that the quality of the design is low. The quality of conformance provides the extent to which the designed specifications are followed during the development of the product. This implies that if the design specifications are set high, but are not followed, then this will affect the quality of the product. Software quality is a mixture of factors that varies with the different software applications.

The factors that affect the software quality are categorized into two groups. They are:

1. The directly measured factors like errors, lines of code, and unit time.
2. The indirectly measured factors like usability or maintainability.

Measurements are considered for both directly and indirectly measured factors. We need to formulate an empirical connection between the quality factors and the software quality itself, which signifies that we understand the indirect factors that determine the software quality.

Following are the indirectly measured factors that affect the quality of software product:

1. ***Reliability:*** Reliability is the extent to which a program performs its proposed function with the required accuracy.
2. ***Efficiency:*** Efficiency is the quantity of computing resources and code required by a program to execute its function.
3. ***Integrity:*** Integrity identifies the extent of control in accessing the particular software product.

4. ***Usability:*** Usability is the effort to describe the user friendliness. It relates to the calculation of total effort required to learn and operate a particular software program.
5. ***Maintainability:*** Maintainability addresses the effort required to locate and fix an error in an operational system, to ensure smooth operation of the system.
6. ***Flexibility:*** Flexibility identifies the effort required to alter a program.
7. ***Testability:*** Testability relates to how far a software program can execute the intended function.
8. ***Portability:*** Portability identifies the effort required to transfer a program from one software to another.
9. ***Reusability:*** Reusability relates to the extent to which a program can be reused in other applications.

All these factors are important in determining the quality of a software product.

Example: Reliability is a software quality factor which cannot be evaluated directly. However, the attributes related to reliability can be measured.

Software Quality Assurance

The function of software quality assurance is to provide a guarantee that the standards, methods, and procedures in place are suitable for the project and can be implemented. SQA (Software Quality Assurance) not just guarantees the use of the recognized standards, processes, and procedures but also gathers various software measures which are necessary for evaluating the standards, processes, and procedures that are being implemented within the organization. Thus, SQA plays a major role in improving the overall performance of the process.

Software Quality Control

Quality Control is defined as "a set of activities designed to evaluate the quality of a developed or manufactured product" (IEEE, 1991); in other words, "Activities whose main objective is the withholding of any product that does not qualify"-- Daniel Galin, in his book, Software Quality Assurance. Therefore, quality control is the inspection carried out after the completion of a product, but before the product is shipped to the client.

Software quality control is a department function, which involves comparing the standards against the product and rectifying the non-conformances detected during the process.

Example: Unit Testing, System Testing, and Integration Testing are software control activities.

Considering the example of testing, software quality control involves operation of a system or application under controlled conditions and evaluating the results against quality goals. The controlled conditions include both normal and abnormal conditions. This consists of a set of methods implemented by companies to ensure that the software product meets the set quality goals at the most excellent value to the client which thereby helps in the progress of an organization.

Quality Assurance is achieved only when quality products are developed. Periodical auditing and reporting systems are the basic tools of quality control. Although, the management has access to data related to the project, it is the 'Audit report', which provides the management the necessary information about the post-inspection data. This provides the management the confidence that the product quality is up to the mark and that it is set to meet the objectives of the developed software.

On developing a new software product, the product variation control is to be set. Variations may occur while designing the product, which in turn affects the quality of the product. By conducting various inspection reviews and investigations, effective control over the quality can be obtained. This verifies that each section of the working module and functionally meets its assigned task. The variations in the process are identified, recorded, and sent back to the development team for correction.

To gain the maximum benefit of quality control system, it is very important that skilled software staff conduct the audit. If the audit report indicates that the software product quality is not up to the mark, then corrective management action is taken accordingly. This helps in maintaining the delivery schedule for the supply of a product to the customer or end-user.

Example:

Consider a software project with a requirement of user interface design and a SQL database execution.

The Software Quality Assurance team would develop a quality plan that contains specified standards, methods and procedures that have to be applied to the software project. They would then involve themselves in identifying and preparing the quality plan that needs to be followed.

When these requirements are produced, the Software Quality Control team ensures that the development team has in fact followed the set standards and presents the audit report to the management.

12.2 Testing and Quality Assurance at Workplace

While developing software, the team members must be aware of terms like software testing, software quality assurance, software quality control, software verification and validation, and software integration and testing. Teams based on these projects and the clients should be made aware of the standards being set for quality of the software being developed. The team members should be made aware of common practices that are being followed at the workplace to accomplish software quality assurance. We will start the same by understanding the difference between software testing and quality assurance.

12.2.1 Differences between Software Testing and Quality Assurance

Quality assurances and testing are two overlapping and confusing terminologies. Though they are closely related, yet they are different. Both quality assurance and testing are necessary to effectively manage the risks of creating and maintaining software products.

Software Testing

Software testing is an essential part of software quality assurance, which denotes a review of specification, design and coding related to software products. In simple words, software testing can be described as an assessment, report, and follow-up-task for accomplishing quality goals. Software testing involves the operation of a system under specified conditions and includes continuous evaluation of the output. These specified conditions include both normal and abnormal conditions. Software testing is mainly detection-oriented and is performed by a software tester.

Following are the tasks performed under software testing:

1. Recognizing the most appropriate implementation approach for a given test.
2. Setting up and executing the tests with the intention of finding bugs.
3. Preparing the verification and validation reports of the test plans, test procedures, and test reports.
4. Involving in customer meetings to know the status of projects and design reviews.

The role of a software tester is to assess, report, and follow-up by tracking down the bugs in the software product and also by ensuring that they are fixed. The main aim of the software tester is to find bugs and ensure that they are fixed as soon as possible.


Notes

Role of the software tester

An effective tester needs to take personal responsibility for the bugs he/she finds, track them through bug life cycle, and convince the software development team to fix it at the earliest.

Quality Assurance

Software quality assurance guarantees that the principles, methods, and procedures are in place, and are suitable for the project and can be implemented properly.


Did you know? Bell Labs in the year 1916 introduced formal quality assurance and control function for the first time. This function was successfully implemented by different companies throughout the world.

Software quality assurance involves different means of preventing occurrence of defects in the software being developed. The various methods that are implemented to ensure the quality are ISO 9000 model or CMM model. The aim of software quality assurance is to continuously improve a clearly defined process and to control every step of the process.

The responsibility of the software quality assurance individual is to inspect and calculate the present software development process and to discover ways to improve it with an intention of preventing the occurrence of bugs. The responsibility and scope of a Software Quality Assurance group is larger than the software testing group.


Example: Consider a User interface standard that has no distinction between an "open new window" and a "replace current window", in the "Go To" button.

During usability testing, it is discovered that, lack of distinction between the "open new window" and a "replace current window", caused the end-user's to search for correct window.

The SQA team sends this information (usability metric) back to the user interface designers and an alteration to the user interface standards are made.

As per the definition of assurance that is "a guarantee or pledge" or "a freedom from doubt," a quality assurance group's role is to guarantee that the quality of the product is good. Quality Assurance (QA) group is authorised to inspect the standards and methodologies followed in a software project. It also monitors and evaluates the software development process to provide feedback solutions to the process problems and performs the testing process to decide whether the product is ready for release in the market.


Example: Mean Time Between Failure (MTBF) is a common software measure that tracks down how often the system fails to perform the given task. This measure, in turn, helps to find out the reliability of the product.

Following are the major quality assurance tasks:

1. To develop a standard process for software development and quality assurance and to ensure that that there is no deviation from the standards set.
2. To set guidelines for every step of the process, such as requirement templates, design methodologies, coding standards, and so on.
3. To create checklists for every step of the process and verify the results of each step against the subsequent guidelines and checklists.
4. To develop quality factors, quality criteria, and quality metrics and define a complete set of quality factors.

Example: Software quality assurance includes checklist and other important specifications related to the software program.

Did you know? Most of the firms have adopted total quality management programs since 1980s to retain competitiveness in order to achieve customer satisfaction in the era of globalization.

Caution Several companies are supporting the "process improvement" mantra as though it is the solution for all their software development and quality problems. However, though a standard process is necessary, it is not adequate. There is no hard evidence that conformance to process standards guarantees good products.

12.3 Quality Management in IT

Quality management, which coordinates the various activities to direct and control the organization with regard to quality, is useful for any industry. Most of the principles and theories of quality management that are applied to software development and maintenance activities can also be applied to all other Information Technology (IT) activities.

The principles and theories that are applied emphasize on the following:

1. To identify the important IT processes and their sequence.
2. To plan for defect prevention versus detection by applying IT best practices.
3. To use and implement various standards to achieve appropriate levels of IT governance.
4. To resolve the IT issues equivalent to bugs, defects, and errors.
5. To determine and document the requirements of the customers.
6. To observe and quantify the service performance.
7. To assure the procurement of quality when outsourcing important IT processes.

The term "Information Technology can be defined as any equipment or interconnected system that is used in automatic attainment, storage, control, and transmission of data. Information technology consists of computers, ancillary equipment, and software."

In order to apply the principles of quality management to the "organized activities" executed by an IT company, it is essential to be familiar with the important IT processes. An IT infrastructure depicts all the components that are utilized in the delivery of the IT services to the end-users, including the computing and telecommunication services. These components and their use should be effectively managed. Therefore, a proper IT infrastructure management should be in place.

The management of IT infrastructure and IT services together is called as IT Service Management (ITSM). ITSM establishes the principles and practices of designing, delivering, and maintaining IT services to an agreed-upon level of quality, with respect to the customer requirement. This section explains the important processes found in a typical IT Company. The processes are quite similar to the software engineering processes found in a software life cycle.

ITSM Processes

IT Service Management (ITSM) is based on two major categories of IT service -- one is IT service support and the other is IT service delivery. The ITSM processes assure that within these two categories, the levels of quality that are agreed-upon are achieved.

Service Support

IT service support consists of processes oriented towards the efficient delivery of IT operational services. The processes are as follows:

1. ***Service Desk Function:*** It serves as a main point of contact for customers and supports the management process in providing the resolution.
2. ***Incident Management:*** It is accountable for maintaining regular position of IT service operations as quickly as possible to minimize the adverse impact on business processes.
3. ***Problem Management:*** It reduces the unfavorable impact of incidents and problems on the business operations caused due to errors in the IT infrastructure and effectively manages the problems that occur.
4. ***Configuration Management:*** It is accountable for recognizing, recording, and reporting on configuration items and their relationships to the supporting IT service department.
5. ***Change Management:*** It organizes and controls all changes to the IT services to reduce the adverse impact of those changes on business operations and the end users of IT services.
6. ***Release Management:*** It implements the various changes to IT services by taking various views of customers, employees, technology, and IT governance that supports all aspects of a change which include planning, designing, building and testing activities.

Service Delivery

IT service delivery includes processes that are associated with the long-term planning, control, and managerial aspects of IT services. The processes include:

1. ***Service-level Management:*** It organizes, plans, coordinates, negotiates, reports, and supervises the quality of IT services at the agreed upon cost.
2. ***Availability Management:*** It is accountable for increasing the ability of the IT infrastructure services and supports organizations to deliver a cost-effective and sustained level of service that meets business requirements.
3. ***Capacity Management:*** It ensures that future and current capacity and performance aspects of the IT infrastructure are supplied on time to meet up business requirements at an agreed upon cost.
4. ***Service Continuity Management:*** It is accountable for business stability management functions by ensuring that IT services are stable even in the event of a major business interruption.
5. ***Financial Management for IT Services:*** It provides budgeting, accounting, and charging services in order to manage and control the IT costs and expenditures.
6. ***Security Management:*** It is accountable for reducing security-related incidents by effectively managing privacy, discretion, integrity, and availability of IT services.
7. ***Applications Management:*** It handles various applications from the initial business setup throughout the stages of application life cycle till retirement.
8. ***Software Asset Management:*** It implements good corporate governance system to manage, control, and safeguard the organization's software assets, as well as those risks arising from the use of software assets.

Example: The major standards established by the ITSM processes are ISO 20000 and CobiT.

Recently, the best practices and standards of IT identified quality as a process and recognized the value of a quality management system.


Task

List out the major quality measurement principles adopted by any major IT industry.


Case Study

Software Quality Assurance

One of the world's major multi-product insurance company called Star Insurance group was facing issues related to its quality and hence identified the need to put into practice the standardized enterprise test metrics to manage its product lifecycle.

After assessment, Star Insurance group developed a test metrics frame work that consisted of several components essential for the success of developing the quality products.

Star Insurance group considered the following factors while designing the metrics:

1. Test metrics for the entire product life cycle management (including the base metrics and metrics collected from the stakeholders).
2. Definitions of each metric.
3. Value of each metrics (This was determined and applied to each phase of the product life cycle).

Preston Company (third party consultant company leader in providing IT related solutions) partnered with the insurance group and was able to develop an interview questionnaire comprising checklist and a rating scale for assessing the current state of the metrics. They had collected views of stakeholders through workshop and daily meetings with the project team to review status issues. Finally they decided upon the frame work of the metrics, implemented a test metric solution and developed the test metrics frameworks to provide knowledge transfer in support of the project.

Preston was able to achieve operational excellence through efficient test metrics. Based on the quality and progressive metrics, project managers were able to take management decisions. Test metrics reduced the risk of schedule overruns on software releases, thereby maintaining the high quality of the application.

Questions

1. Why did Star Insurance Company identify the need for standardized enterprise test metrics?
2. What was the solution suggested by Preston company?

Adapted from http://www.questcon.com/documents/documents/1/EffectiveTestMetrics.pdf

12.4 Summary

- Quality is defined as the features or the attributes of the products that are appreciated by the end-users or the customers. Quality assurance is an orderly procedure of inspecting if a particular product or a service that is being developed is meeting the required standards.
- A good quality assurance system increases an organization's credibility and belief as well as improves work processes and efficiency that enables the organization to compete with others.
- When a software system or process reaches a particular standard and this standard satisfies the customer expectations, then it is said to be a quality software system or process. The factors that affect the quality of the software products are efficiency, integrity, usability, maintainability, and flexibility.
- Software testing is an essential part of software quality assurance and denotes a review of specification, design, and coding, where as software quality assurance involves different means of

inspecting the software engineering processes and various methods that are implemented to ensure quality, such as the ISO 9000 model or CMM model.

- Information Technology can be defined as any equipment or interconnected system that is used in the automatic attainment, storage, control, and transmission of data. Information technology consists of computers, ancillary equipment, and software.
- The management of the IT infrastructure and IT services is called as IT service management (ITSM). ITSM establishes the principles and practices of designing, delivering, and maintaining IT services to an agreed-upon level of quality, with respect to the customer requirement.
- IT Service Management (ITSM) is completely based on two major categories of IT service -- one is IT service support and the second one is IT service delivery.

12.5 Keywords

Ancillary Equipment: Support providing equipment.

Datum: Fact or a principle. .

Inspection: A group review on quality for written material.

Quality Audit: A systematic examination to determine whether the activities and results fulfill the set quality agreement.

12.6 Self Assessment

1. State whether the following statements are true or false:
   (a) Software quality is a mixture of factors that remains constant across various software applications.
   (b) Most of the principles and theories of quality management that are applied to a software development and maintenance activities cannot be applied to IT related activities.
   (c) A test group frames quality assurance policies and objectives in an organization.
2. Fill in the blanks
   (a) _____ is an orderly procedure of inspecting a particular product or a service that is being developed in meeting the required quality standards of the organization.
   (b) Periodical auditing and reporting systems are the basic tools of_____.
   (c) _____ serves as a main point of contact for customers and supports the management process in providing the resolution.
3. Select the suitable choice for every question.
   (a) Which of the quality factor identifies how far a software program executes its intended function?
      (i) Testability (ii) Portability (iii) Reusability (iv) Flexibility
   (b) Which among the following guarantees that the principles, methods and procedures that are in place are suitable for the project and can be implemented properly?
      (i) Software quality control (ii) Software quality assurance
      (iii) Software testing (iv) Software metrics
   (c) Which of the following ensures that the performance aspects of the infrastructure are supplied on time to meet up business requirements at an agreed upon cost?
      (i) Problem management (ii) Capacity management
      (iii) Change management (iv) Configuration management

(d) Identify which of the following tasks is not performed under Software Quality Assurance.

(i) Developing a standard process for software development to ensure that there is no deviation from set standards.

(ii) Setting guidelines for every step of the process such as templates, design methodologies and coding standards.

(iii) Creating checklists for every step of the process to verify the results of each step against the subsequent guidelines.

(iv) Recognizing the most appropriate implementation approach for a given test.

12.7 Review Questions

1. "Even though software testing is considered to be an essential part of software quality assurance, much importance is given to Software Quality Assurance." Justify.
2. "Software quality is a mixture of factors that varies across different software applications." Analyze the various factors that affect the quality of the product.
3. "ITSM processes provide assurance with the help of its two categories called service support and service delivery." Discuss.
4. Do you think the software quality control different from general quality control which is applied in almost all industries? How?
5. "To develop a standardized process for software development is one of the tasks performed under quality assurance." Briefly discuss the other important tasks.
6. "The quality management, that coordinates various activities to direct and control the organization with regard to quality, can be useful for any industry." Do you agree? Justify.
7. "Quality means conformance to requirements." Briefly explain the terms quality, quality control and quality assurance.
8. "IT service delivery includes processes that are associated with the long-term planning, control, and managerial aspects of IT services." Discuss.

Answers: Self Assessment

1. (a) True (b) False (c) False
2. (a) Quality assurance (b) Quality control (c) Service desk function
3. (a) Testability (b)Software quality assurance (c) Capacity management

(d) Recognizing the most appropriate implementation approach for a given test

12.8 Further Readings

Schulmeyer Gordon. G. (2008), Hand book of software assurance-fourth edition. USA. Artech House

GALIN, Daniel (2006), Software Quality Assurance, UK, Pearson Addison Wesley.

http://www.mosaicinc.com/mosaicinc/rmThisMonth.asp link

http://flylib.com/books/en/4.223.1.170/1/-organisation Structure

Unit 13: Quality Management in Organizations

CONTENTS

Objectives

After studying this unit, you will be able to:

- Describe the test management and organizational structure
- Explain software quality assurance metrics

Introduction

We are aware of the fact that software testing alone cannot guarantee a product's quality. An organization strives to improve the quality of the product through various ways. Firstly, organizations institute various standards and methodologies of software development. Then the development of the software are carefully and methodically monitored and evaluated. There might be several problems faced during development which are corrected and then a methodology to prevent such errors is also taken care of. It is after this process that the software is tested. Organizations hence have a quality assurance group to achieve their goal of quality control. Total Quality management approach has great ramification in this regard, which creates a quality culture in organizations. The quality culture thus permeates into the entire organization and includes the grass-root developer who creates the foundation of the software product.

The contribution of software quality metrics is to assess whether the process of development follows the software quality requirements. The quantitative analysis that is brought about by software quality metrics brings clarity, thereby reducing subjectivity in the assessment of software quality. The use of metrics helps in analysis of software projects and does not eliminate the human judgment. It makes software quality more visible. We would discuss the various factors that affect software quality metrics in this unit.

13.1 Test Management and Organizational Structure

Test management is an important part of software quality. It is the practice of categorizing and controlling the entire software process. The goal of test management is to allow software teams to develop, execute, and evaluate all the testing activities within the overall software development. In addition to test management, the test group's name and its implicit responsibilities deeply affect and fit into the organization's overall management structure.

The organizational structure is the hierarchical arrangement within an organization, which organizes its line of authority and flow of communications. There are various organizational structures available. Each has its positive and negative features. Some of the common structures are discussed below.

Example: Figure 13.1 depicts the organizational structure for a small project, it is noticed that test team always reports to the development manager.

Figure 13.1: Organizational Structure for a Small Project

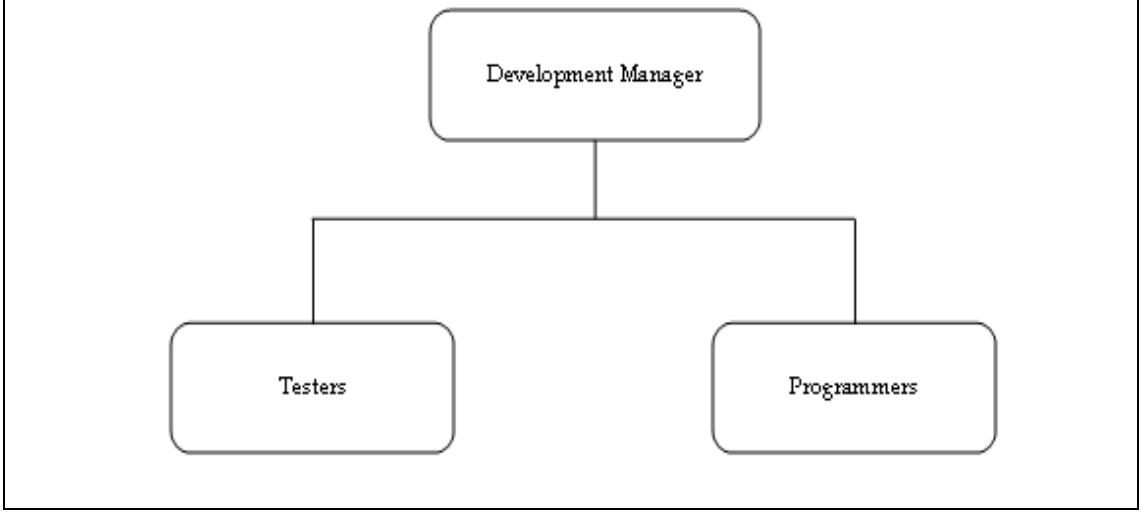


Figure 13.1 shows an organizational structure used frequently by small project teams. As per this figure, the test group always reports to the Development Manager, the person managing the work of the programmers.

The main aim of the development manager is to motivate the team to develop software. At times the software process can come across small hindrances when bugs are reported by testers. In spite of many negatives, this structure works efficiently, if the development manager is experienced and is conscious that his/her aim is not only to motivate the team to create software, but also to create quality software without any bugs. In the presence of such managers, testers will be valued equal to the programmers. The above structure is said to be ideal for flow of communication. If there are minimal layers of management, then the testers and programmers can work together efficiently.

Example: Figure 13.2 depicts organizational structure in which the test group as well as the development group report to the manager of the project.

Figure 13.2: Organizational Structure with Test Group and Development Group.

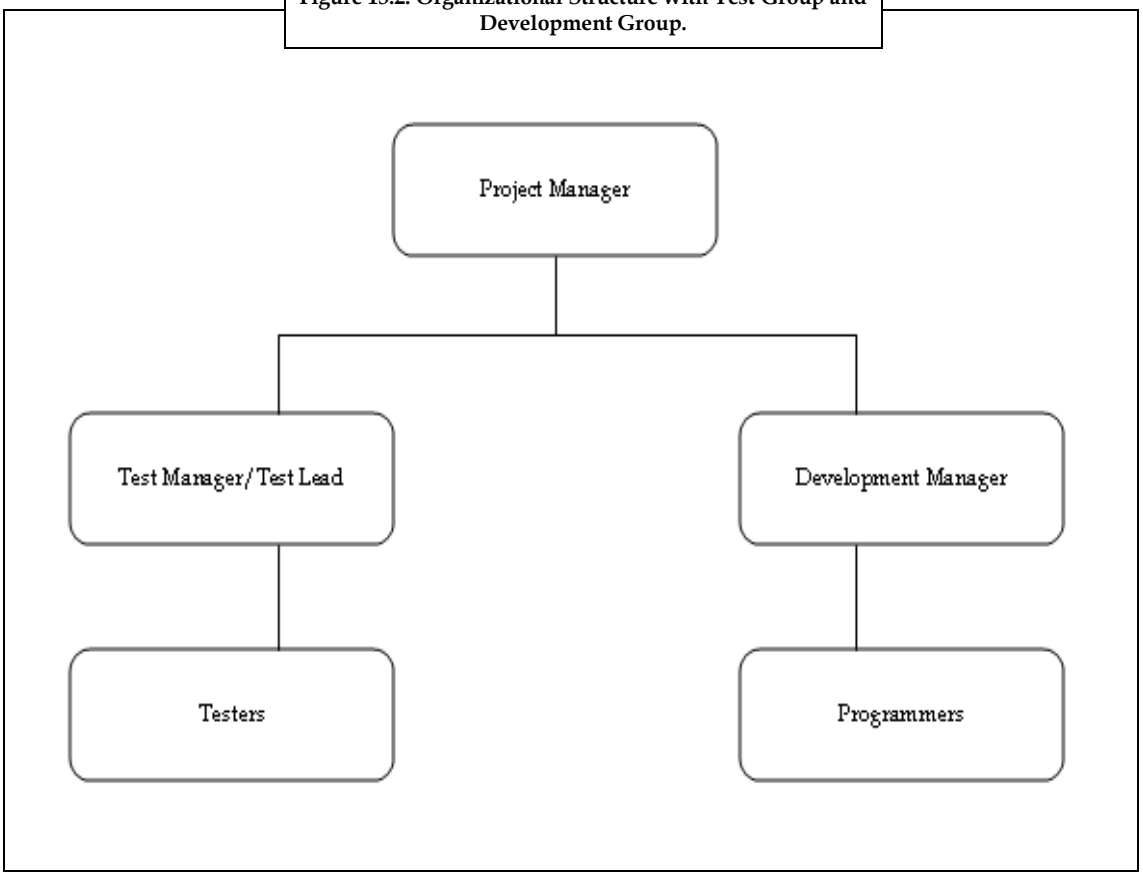


Figure 13.2 depicts the organizational structure wherein both the test group and development group report to the manager of the project. In this organizational structure, the test groups have their own team lead or manager. The manger or the team lead focuses on the test team and their work. The opinions of both the test team lead and the programmer are given equal importance. This provides a great advantage when critical decisions are made regarding the software's quality.

Even though the testers' and programmers' views are considered for decision making, ultimately it is the manager who decides on the quality of the product. This kind of approach may be suitable for a few industries. But in case of high-risk or mission-critical systems, it is always better to have the voice of quality heard at a higher level.

Example: Figure 13.3 depicts the quality assurance or test group that reports to executive management

Figure 13.3: Test Group Reporting to Executive Management

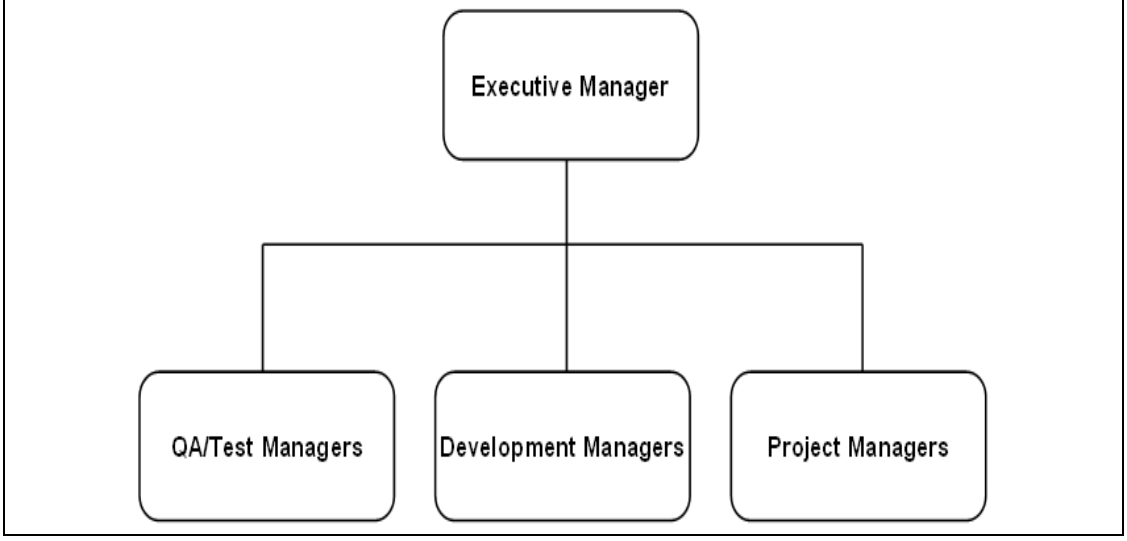


Figure 13.3 depicts the quality assurance team or test group reporting to the executive management. In this kind of organizational structure, the teams accountable for software quality directly report to the senior management, based on their individual projects. The level of authority is equally segregated between the quality assurance level and the testing level. The teams are allowed to set standards and guidelines, measure the results, and adopt various processes to improve the software quality. Any information or report concerning quality of the product is directly sent to the top management.

The group is independent of the project, but this does not mean that they can set unreasonable and difficult quality goals, unless and until it is the requirement of the project. A corporate quality standard that works well on database software may not be suitable when applied to a computer game.

To be effective, an independent quality organization has to find different ways to work with all the projects they deal with, and set the standard for quality with the practicality of releasing the software.

These three organizational structures are just a few examples. The positives and negatives discussed for each structure may differ widely. In software development and testing, one size doesn't necessarily fit all; neither does the standard that works for one team will be suitable for another one.

13.2 Software Quality Assurance Metrics

The basic difference between a metrics and an indicator is that metric is a certain rule that is used to measure some features or trait of a computer software entity, where as an indicator is defined as a variable that can be set to a prescribed state based on the results of a process. The main aim of the software quality metrics is to evaluate whether the software quality requirements are being met throughout the development cycle of the software product. The metrics that we obtain from the software serves as a base for software testing and design decisions. Once a software product is delivered to the end user, we measure the number of defects found during the maintainability of the system using software quality metrics. Software quality assurance metrics are closely connected with software development metrics.

Example: File size metric. It is used to determine the total number of characters in the main files of a program.

Quality metrics play an important role in statistical quality assurance. The defects that are observed during the maintenance phase are recorded and detailed information about the cause of the defect is identified, so that the right corrective action can be taken. Metrics also helps in identifying defect trends, thereby helping to plan for their prevention.

Metrics are required to know the health of the process and the project in an organization. Metrics actually check whether the quality standards set by an organization for a particular project is achieved in a systematic way or not.

There are two types of metrics, namely:

1. ***Product Metrics:*** Product metrics usually measure the readiness and completeness of the software.

Example: Product metrics measures the final size of the program or the number of pages documented in a particular stage of software development.

2. ***Process Metrics:*** Process metrics measure the overall software development process.

Example: Time taken for overall development of a product, type of methodology used, or the average level of experience of the programming staff.

Different organizations follow different dimensions to measure the software quality of their product. The most common software metrics followed by organizations are:

1. Code Coverage
2. Bugs Per Line of Code
3. Cyclometric Complexity
4. Function Point Analysis
5. Number of Classes and Interfaces
6. Cohesion
7. Coupling
8. Order of Growth
9. Source Lines of Code

Example: The metrics used to measure the complexity of a software module which have been already developed are the McCabe's cyclometric complexity measure and the Halstead's measure of conciseness.

The McCabe's cyclometric complexity measure is used to measure the complexity of a software module, as it is very difficult to rewrite the software program that has been already completed. To have a better understanding about the complexity of measures, rules are created to ensure low complexity of the programs during the design and coding phase.

Software Quality Indicators

Management concerns can also be addressed with the help of quality indicators. Some of the recommended quality indicators are:

1. ***Progress:*** It measures the amount of work done by the developer in each phase.
2. ***Stability:*** It evaluates whether the products of each phase are adequately stable to proceed to the next phase.
3. ***Process Compliance:*** It measures the developer's obedience with the development procedures approved during the start of the project.

4. ***Quality Evaluation Effort:*** It measures the developer's effort that is being spent on internal quality evaluation activities.
5. ***Defect Detection Efficiency:*** It measures how many defects were detected in a particular phase.
6. ***Defect Removal Rate:*** It measures the total number of defects detected and resolved over a period of time.
7. ***Defect Age Profile:*** It measures the total number of defects that have not been resolved over a period of time.
8. ***Defect Density:*** It identifies the defect-prone parts of the system.
9. ***Complexity:*** It measures the complexity of the code in a particular program.

Did you know? Scientific Systems, Inc. along with the Air Force Business Research Management Center developed software quality indicators to improve the management capabilities of personnel responsible for monitoring software development projects.

The important measure for a project's success is quality. Products with high quality would result in customer satisfaction, while poor quality products result in customer dissatisfaction. Quality measures should help in identifying the problems, and suggesting corrective actions which must be implemented. They should not be time consuming or dependent on extensive software training. The quality measures should be flexible, clearly defined, easy to calculate, and straightforward to interpret.

Did you know? Hewlett Packard (HP) was a leader in software quality metrics.

Task

Prepare a list of software quality metrics that can be used for measuring the quality of software.

13.3 Summary

- The contribution of software quality metrics is to assess if the development process follows the software quality requirements. The quantitative analysis reduces subjectivity in the assessment of software quality.
- There are various organizational structures available, each of them having its own positives and negatives in it. Some of the common examples are organizational structure for a small project with test team always reporting to the development manager or the quality assurance team or test group reporting to the executive management.
- Metrics are certain rules used to measure some features or traits of a software entity .The main aim of the software quality metrics is to evaluate whether the software quality requirements are being met throughout the development cycle of the software product.

13.4 Keywords

Code Coverage: An analysis method that determines which parts of the software have been executed and which part is still remaining by the test case suite.

Coupling: The extent to which each program module depends on the other module to function effectively

Cyclometric Complexity: It is one of the software metrics used to indicate the complexity of the program.

Function Point Analysis: Function Point Analysis is a reliable metrics that helps in estimating projects, managing change of scope, measuring productivity and communicating functional requirements.

13.5 Self Assessment

1. State whether the following statements are true or false:
   (a) Process compliance measures the developer's obedience with the development procedures approved at the beginning of the project.
   (b) Metrics actually check whether the quality standards set by an organization for a particular project is achieved in a systematic way or not.
   (c) Quality measures should be fixed and rigid.
2. Fill in the blanks
   (a) A _____ quality indicator identifies the defect-prone parts of the system.
   (b) _____ serves as a base for taking decisions related to software testing.
3. Select the suitable choice for every question.
   (a) Which of the software quality indicator identifies developer's obedience with the development procedures approved during the start of the project?
      (i) Process compliance
      (ii) Defect density
      (iii) Complexity
      (iv) Defect detection density
   (b) Which of the following measures the total number of defects detected and resolved over a period of time?
      (i) Defect density
      (ii) Defect age profile
      (iii) Defect detection efficiency
      (iv) Defect removal rate

13.6 Review Questions

1. "Organizational structure is the hierarchical arrangement within which an organization organizes its line of authority and flow of communications." Elaborate.
2. "Management concerns can be addressed with the help of quality indicators." Explain.
3. Are quality metrics and quality indicators one and the same? Explain the difference
4. "Different organizations follow different dimensions to measure the software quality of their product". Discuss the most commonly used software quality metrics.
5. "Quality metrics play an important role in statistical quality assurance." Substantiate.
6. "The important measure for a project's success is quality." How is quality and customer satisfaction interrelated?

Answers: Self Assessment

1. (a) True (b) True (c) False
2. (a) Defect density (b) Metrics
3. (a) Process Compliance (b) Defect age profile

13.7 Further Readings


Books

Schulmeyer Gordon. G. (2008), Hand book of software assurance-fourth edition. USA. Artech House

GALIN, Daniel (2006), Software Quality Assurance, UK, Pearson Addison Wesley.


Online link

http://www.mosaicinc.com/mosaicinc/rmThisMonth.asp link

http://flylib.com/books/en/4.223.1.170/1/-organisation Structure

Unit 14: Maturity Model and Quality Standards

CONTENTS

Objectives

After studying this unit, you will be able to:

- Explain CMM.
- Describe the elements of ISO 9000.
- State the various software engineering standards.

Introduction

The goal of every organization is to achieve sustainable excellence in its operations. Every organization should have a proven framework to improve the performance of their information technology systems to meet their current requirements. It should be flexible enough to adjust to their varying business needs. However, it is a challenge to achieve this in the face of global competition, rapid technological innovation, and changing customer demands. This is because most organizations work with poorly designed IT applications, operate with high budget, and are often late to deliver their projects.

Quality assurance is a process by which the organizational structure of an organization can be defined. Quality assurance methods can be implemented through the application of maturity models and systematic procedures. A maturity model helps the organization to develop and support its information systems. It also helps the organization to accomplish its work with high quality and low cost. The maturity model helps to control complexity of today's huge systems. CMM is one such model which will be discussed in detail in this chapter.

Software Quality Assurance through quality management system is a process-driven approach with specific steps to attain goals. A quality management system gives out the framework that allows an organization to assess and improve process capability, manage risk effectively, and achieve customer satisfaction and loyalty. It is a big challenge in this world to achieve this state, considering the present day's situation of the IT industry.

A quality management system is a realistic and sensible method, which promotes a methodical approach towards the development of product. They make products and services well-organized through persistent improvement, enabling international recognition and patronage. These standards make sure that popular characteristics of products and services bring in safety and reliability to products thereby improving customer satisfaction. ISO 9000 is one such standard which will be discussed in detail in this chapter.

14.1 CMM (Capability Maturity Model)

The Capability Maturity Model (CMM) is a standard model used for depicting and measuring the maturity of a software company's development process. CMM model also provides guidance on how companies can improvise their software quality. It was developed by the software development community along with the Software Engineering Institute (SEI) and Carnegie Mellon University under the direction of United States Department of Defense. CMM is unique in its manner because it is incremental in nature and can fit into any class of a software company, ranging from a startup company to a well-established company.

CMM defines five-level steps for process mapping and implementation. These steps represent a model which explains the process maturity of an organization and analyzes the current state of process maturity of an organization. Organizations make use of these steps to assess their maturity levels before skipping onto another level. This flexibility to slowly strive towards high process maturity helps the organizations to easily adopt to process related changes. They can go from first level to fifth level over a period of time, and need to go from one level to another. This implies that they cannot skip any level.

The Capability Maturity Model (CMM) is basically meant for software development organizations. It caters to the field of software engineering, system engineering, project management, software maintenance, risk management, system acquisition, information technology (IT), and personnel management.

Did you know? It has been estimated that since 1987, the number of companies using CMM to assess their software management practices has doubled every five years.

CMM is a widely used and preferred software method of evaluation. It involves development of software operating measures, which are developed in five step quality conditions ranging from CMM1 to CMM5. CMM contains various levels and structures. Let us first understand the structure and components of CMM and then get more details on the five levels.

Notes

A Maturity Model:

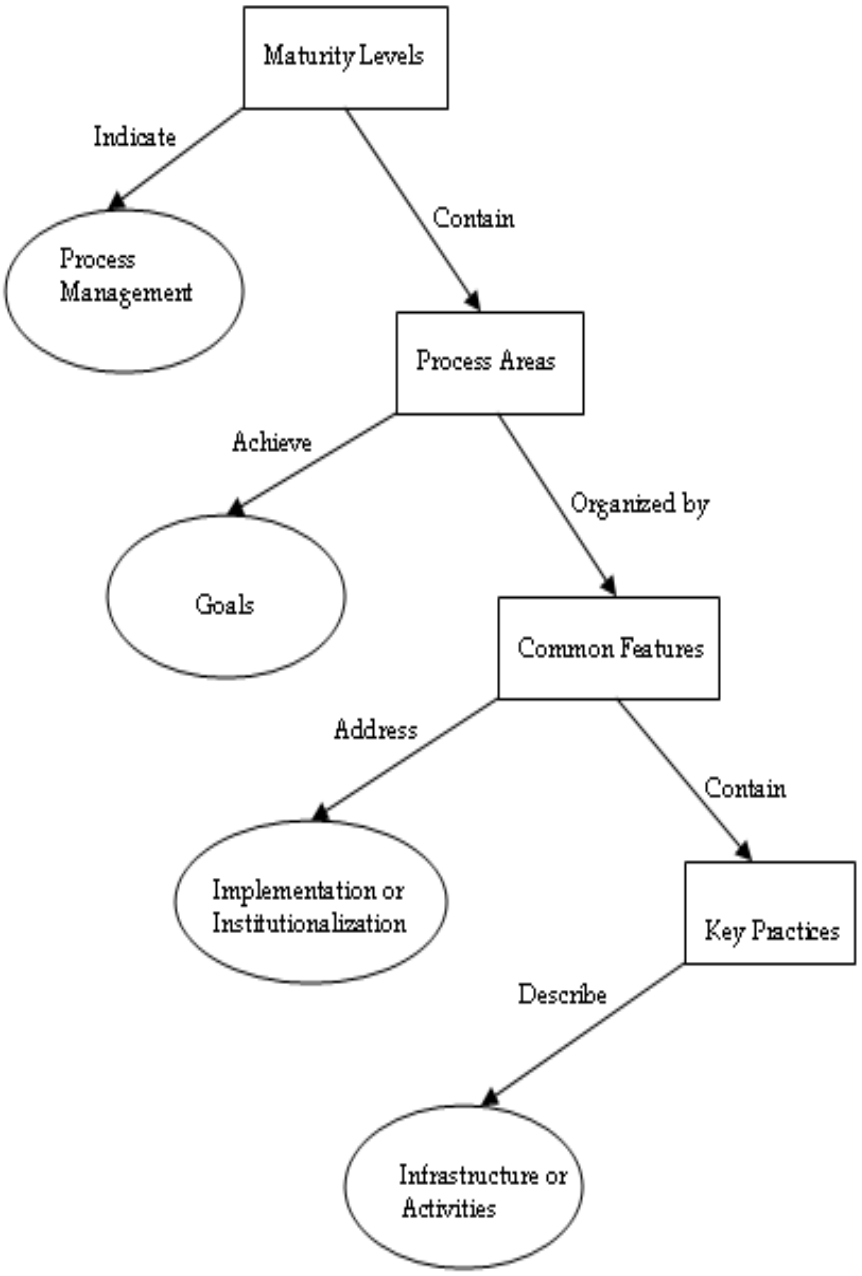
1. Is a combination of structured levels that showcases how efficiently processes, practices, and behavior can produce the desired outcomes.
2. Can be utilized as a model for evaluation and as a guide for analysis.

CMM Structure and Components

Figure 14.1 illustrates the structure of CMM. Let us understand how the CMM components are interwoven in its structure. CMM has maturity levels that indicate the process capability at any point of time. These maturity levels contain process areas. Each level has different process areas. These process areas have some goals to achieve and are made up of common features or characteristics. These common features are meant to address implementation or institutionalization aspects of process improvement. The common features contain some key practices which are followed as a main guideline for process improvement.

Figure 14.1 shows the CMM structure and its components.

Figure 14.1 CMM Structure and Components



Let us now discuss in brief the main components of CMM.

1. ***Maturity Levels:*** The maturity levels represent the level of process capability an organization possesses. CMM has five levels of maturity. Here, the top level is an ideal state where processes are systematically managed through process optimization and continuous process improvement, while the initial level is characterized by ad-hoc processes.
2. ***Process Capability:*** This indicates if the current state of processes will help the organization in meeting its quality expectations. This capability helps in predicting the results for an upcoming software project that the organization takes up.
3. ***Key Process Areas (KPA):*** It is a group of associated activities that strives to attain a set of goals when performed together. It also sets up a process capability at the maturity level. For example, Software project delivery planning.
4. ***Goals:*** It reviews the essential practices of a process area and tells whether a project or an organization has been successful in implementing the process area. The goal also points to the importance and the purpose of each process area.
5. ***Common Features:*** Common features are the characteristics that tell whether the implementation of a key process area is successful and permanent.
6. ***Key Practices:*** Each process region is defined in terms of key practices that assist to help the key process area. The key practices describe the infrastructure and performances that matter most to the application of a process. For example, the project's software delivery schedule plan is designed according to a documented procedure.

Reading further will help us to understand the five maturity levels of CMM.

14.1.1 Five Levels of CMM

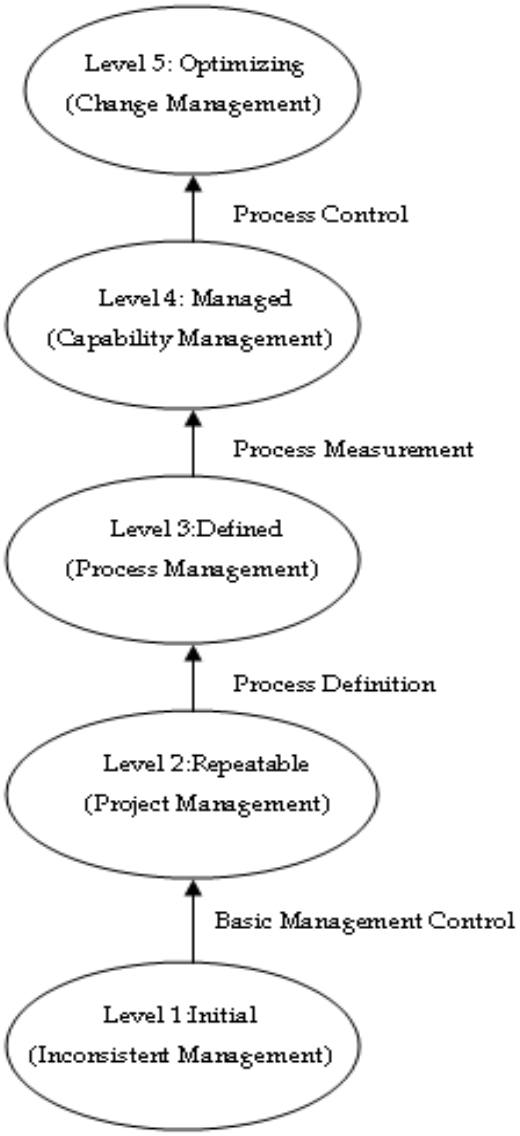
CMM for software has an extensively accepted set of strategies for developing high performance software companies. Improvised set of software development methods cannot sustain unless Application Development (AD) company behavior changes to help them. Humphrey planned the process maturity framework to assist AD companies and to increase the quality of their AD methods in five levels.

Basically made up of five levels of development, CMM is an exclusive model which depicts an organization's growth and change. The five level development methods are changed from a disordered state to an ordered state, which is capable of producing good results.

As an AD organization advances from one level to another, its depiction changes, and it undergoes improvisation of development processes. This has been illustrated in figure 14.2.

Figure 14.2 shows the CMM levels.

Figure 14.2 CMM Levels



Every maturity level is shown by the implementation of various groups of process areas that assist the developmental ability acquired at that level. According to SEI, "Predictability, effectiveness, and control of an organization's software processes are believed to improve as the organization scales up to these five levels. While not rigorous, the empirical evidence to date supports this belief." The five maturity levels are as described below:

1. ***Level 1- Initial:*** The starting phase for using a new process. At this level, the processes are not well structured and are in a state of rapid change. These processes are intended to travel in an uncontrolled, reactive manner by users or events. This makes way for an uncomfortable environment for the processes. At the ground level, AD practices and results are not consistent. Growth processes are not defined clearly, and good practices are often not taken care to meet tight schedules. There is lack of consistency in project management at this level. This will not

protect developers from the ridiculous commitments or immediate requirement changes. Level 1 organizations generally fail to meet their goals in a consistent manner.

2. ***Level 2 – Repeatable:*** In this level, the process is managed according to the metrics described in the initial level. Software development goals are repetitive in nature. It is apt to study an environment which intends to repeat successful procedures. The procedures might not be repeated or be the same in the organization. The organization can make use of some fundamental project management practices to minimize cost and schedule. Thus, Level 2 aims to develop the capabilities of project managers to:

   (i) Attainable project plan

   (ii) Control of project requirements

   (iii) Managed product modifications

   Project can make use of variety of procedures for developing software. But, a stable environment to support the software is necessary for better performance. Companies with level 2 potentials submit their deliverables on time. In a process, discipline helps them to ensure that efficient practices are followed when tight timelines are scheduled. When these practices are consistent, projects can be well executed according to the documented plan. Project status and release of deliverables are noted by the management at regular intervals. The performance of any project becomes repeatable at this level.

3. ***Level 3 – Defined:*** This level is marked with well defined processes. The foundation for level 3 is the organization's set of standard procedures, which are recognized and enhanced over a period of time. These standard processes are used to determine consistency across the organization. The organization's management defines the process objectives based on the organization's set of standard processes and ensures that these objectives are properly managed. Project teams set up their distinct processes based on the organization's set of standard processes and strategies.


Notes

A considerable variation between level 2 and level 3 is the prevalence of standards, process descriptions, and events:

1. At level 2, the standards, processes, and events may differ in every occurrence of the process.

2. At level 3, the standards, process descriptions, and events for a project are transformed from the organization's set of standard processes to match a particular project or an organizational element.

After projects reiterate flourishing methods, organizations recognize these methods from different projects. Therefore, these events are integrated into a general process and arranged across the organization. Hence, a strong organizational culture shows up at level 3, which is based on a general process that covers all the important elements.

4. ***Level 4 - Quantitatively Managed:*** Software development is effectively controlled by taking correct measurements in management of projects; the management can adapt to fit in these processes without bringing any loss in the quality for software processes and software maintenance. Sub processes that significantly include overall process performance are selected. These selected sub processes are based on the use of statistical and other quantitative techniques.

   At level 3, procedures are predicted qualitatively. After similar processes are established, organizations then can develop baselines with certain potentials that predict certain results from performing these processes. By managing the performance of its growth processes statistically, an organization can forecast and control project outcomes much earlier in the course of a project.

   These baselines provide deep, quantitative analyses of the capability of development processes and the causes of difference in their performance. A significant difference between level 3 and 4

is the certainty of process performance. At maturity level 4, the process performance is managed using statistical and other quantitative methods, and is conventional in nature.

5. ***Level 5 – Optimizing:*** At this level, process management includes deliberate process optimization/improvement. This level stresses on repeatedly getting better with process performance through incremental and creative technological improvements. Both the defined processes and the organization's set of defined processes are targets of measurable improvement performance. The effects of organized process improvements are measured and assessed against the quantitative process-improvement objectives. Quantitative process-improvement objectives for the organization are recognized, frequently revised to reflect changing business objectives, and used as criteria in managing process improvement.

   At level 5, processes are associated with resolving common causes of process differences and changing the process (that is, shifting the mean of the process performance) to improve process performance (while maintaining statistical probability) to achieve the established quantitative process-improvement objectives. On the other hand, at level 4, developments are associated with special causes of process variation and providing statistical predictability of the results. Though processes may produce expected results, the results may be inadequate to attain the recognized objectives.

Table 14.1 gives an overview of the different maturity levels and the improvements implemented in each of these levels.

Table 14.1 Capability Maturity Model Process Areas

Maturity Level	Improvements Implemented
5. Optimizing	Develop change infrastructure Assess and organize improvements Remove causes of defects
4. Quantitatively Managed	Supervise processes quantitatively Create capability baselines
3. Defined	Create enhancement infrastructure Identify required software processes Organize and manage processes Collect process-level data Provide organization-wide training Manage with non-software groups
2. Repeatable	Supervise requirements Plan and track projects Manage suppliers Manage product configurations Measure projects Assist and assure policy compliance
1. Initial	No required processes

Within each of the maturity levels are Key Process Areas (KPA) which explain about that level, and for each KPA there are five aspects, namely:

1. ***Goals:*** Each KPA sets several goals which include the different objectives that help the organization to climb to the next CMM level.
2. ***Commitment:*** There are various requirements that must be met in order to achieve the goals that are specified.
3. ***Ability:*** The abilities define the activities carried out to enable the organization meet the commitments.
4. ***Measurement:*** The activities performed should be monitored, and hence there are methods that have to be followed to implement the same.

5. ***Verification:*** There are various methods that need to be followed that defines the verification.


Caution

A company wanting to go to the topmost CMM level must have implemented the previous four levels.

Benefits of CMM

Let us now list the benefits of CMM.

1. CMM is simple and easy to analyze.
2. Many corporate executives are already familiar with the CMM model.
3. Many companies successfully use the CMM model to illustrate major IT issues/concepts to senior executives.
4. CMM model is a valuable tool for attaining project funding for key initiatives -- for example, enterprise data asset management and Meta data repository development.
5. Many large companies and government institutions are actively using the CMM model to compare themselves with other entities.

CMM is now advancing with the introduction of Capability Maturity Model Integration (CMMI). CMM describes software engineering only whereas CMMI describes integrated processes and disciplines, and hence applies to both software and systems engineering. It was brought out in 2002 with the idea of improving the usage of maturity models in software engineering. CMMI contains 22 process fields separated into four categories, where maturity levels are defined for each. Of the 22 process goals, every goal consists of different practices. It assists the organization to evaluate the current processes and also provides an organized way to improve these processes.


Did you know?

Wipro Technologies, Cognizant Technology Solutions , Infosys Technologies Limited, Larsen & Turbo Infotech Limited, Mastek Limited, NIIT, Software Solutions, Patni Computer Systems Limited, Satyam Computer Services Limited, Sonata Software Limited, Syntel , Siemens Information Systems Limited., Tata Consultancy Services, Tata Elxsi Limited, Software Paradigms International (SPI) are some examples of CMMI companies.

14.2 ISO 9000

ISO 9000 is a written set of standards published by an international standard writing body (International Organization for Standardization). The rules describe practices that are universally recognized and accepted for assuring that organizations perform well and understand the requirements of clients. ISO is highly generic in nature. Its standards can be applied to any kind of an organization, providing any kinds of services or product.

"ISO" is derived from the Greek word isos, which means "equal." The main goal of ISO 9000 is to establish a standard interpretation of quality assurance throughout the world.


Notes

Two theories as to how the name ISO emerged.

1. One theory is that ISO relates to the two equal sides of an Isosceles triangle and International Organization of Standards is consistent and equal right through the international community.
2. Another theory has it that the French insisted on the French version of International Organization of Standards which is pronounced in the order of "International Standards Organization", leading to the abbreviation of ISO.

There are 20 crucial elements in ISO 9000 standard. It explains the measures that a company has to follow in order to show that it is carrying out all its procedures as per the ISO 9000 rules. Each of these elements should be clearly and completely documented.

1. ***Management Responsibility:*** The management of the organization should set the quality policy and implement the same by providing resources, personnel and training.
2. ***Quality Systems:*** The Quality system comprises of a Quality manual which describes the Quality processes and has supporting procedures that are created and maintained.
3. ***Contract Review:*** The needs and expectations of the customer are explicitly stated in a Contract review.
4. ***Design Control:*** There are various design elements like Engineering drawings, the changes of which have to be carefully documented to ensure that the changes have been fully coordinated and approved internally.
5. ***Documentation and Data Control:*** The creation and modification of documents that support the Quality system is controlled strictly by the ISO procedures.
6. ***Purchasing:*** Procedures for purchasing describe the requirements of raw material suppliers and also establish the processes for ensuring compliance to the standards are developed.
7. ***Control of Customer Supplied Product:*** Procedures that detail out methods of handling the raw material supplied by customer and its safekeeping are developed.
8. ***Product Identification and Traceability:*** Products at all stages (start to finish) have to be tracked and methods for tracking the dates are maintained to enable traceability.
9. ***Process Control:*** Process control is established through work instructions, quality plans and workmanship standards that verify whether each job is being done correctly.
10. ***Inspection and Testing:*** There are three stages of Inspection, namely receiving, in-process and final inspection areas that are inspected and tested for quality. The records of inspection are preserved as a part of quality system.
11. ***Inspection, Measuring and Test Equipment:*** The instruments that are used should be calibrated regularly and records should be maintained.
12. ***Inspection and Test Status:*** Inspected materials may be used or processed further. An Inspected product should be easily identified.
13. ***Control of Non Conforming Product:*** Materials or products that fail the specifications are rejected and should be separated from the approved production. A process should be in place to handle the rejects – whether they should be reworked or returned to the supplier.
14. ***Corrective and Preventive Action:*** Corrective action focus on identifying the root cause of the problem and Preventive actions define the measures to ensure that defects do not happen in future.
15. ***Handling, Storage, Packaging, Preservation and Delivery:*** The finished product should have procedures that outline practices that protect products from damage during manufacturing and shipping.
16. ***Quality Records:*** Quality records are required to provide an audit trail that will be used for internal and external auditors.
17. ***Internal Quality Audits:*** It is essential that quality processes are regularly (once in a quarter, half-yearly) audited internally to ensure that all elements of ISO are followed. The records of internal quality should be shared with the external auditor.
18. ***Training:*** Training records are maintained that show the levels of expertise of every employee.
19. ***Servicing:*** Servicing refers to the support of the product after delivering to customer. If servicing is specified in the contract, procedures should be established that verify that servicing meets the indicated requirements.

20. ***Statistical Techniques:*** Process correctness is governed by control charts, graphs and other methods of analysis. The reports are also used for continuous improvements.

For an organization to perform well, it has to integrate these twenty elements at the right time and under the right circumstances to produce an efficient system. This is usually called a quality system, but a lot of business analysts refer to it as a way of doing business, or a management system.

Quality Management Principles in ISO

This section speaks about generic descriptions of principles as explained in ISO 9000. In addition, it provides examples of the benefits from their usage and actions that managers typically take in applying the principles to improve their organizations.

1. ***Principle 1 - Customer Focus:*** Customers are the main target sector of all organizations and therefore they need to understand current and future customer needs, should meet customer requirements, and strive to exceed customer expectations.

   The advantages associated with customer focus are:

   (a) Organizations can increase the revenue and market share as a result of flexible and rapid responses.

   (b) Organizations can utilize their resources effectively to improve customer satisfaction.

   (c) Organizations will be able to improve loyalty, thereby leading to repeat business.

   Following steps need to be taken by organizations to apply this principle:

   (a) Organizations to focus on customers primarily, that is, they research, understand and communicate with customer to find out their needs and expectations.

   (b) Measuring customer satisfaction and performing on the results.

   (c) Managing customer relationships effectively.

   (d) Adopting a balanced approach to please customers and other concerned parties such as owners, employees, suppliers, financiers, local communities and the society as a whole.

2. ***Principle 2 – Leadership:*** Leadership plays a very important role in achieving an organization's goals. Leaders are responsible for creating and maintaining the internal environment in which employees can become fully involved in achieving the organization's objectives.

   The advantages associated with this principle are:

   (a) Employees can be taken under trust and be encouraged to achieve organization's goals and objectives.

   (b) Activities are shaped and organized in an integrated way.

   (c) Minimal misunderstanding between the different levels of an organisation.

   After applying this principle:

   (a) The necessity of all concerned parties including customers, owners, employees, suppliers, financiers, local communities and society can be taken into consideration.

   (b) Required objectives and targets can be set to ensure a clear vision of the organization's future.

   (c) Initialization and maintenance of common values, equality and inspiring ethical role models can be established at all levels of the organization.

   (d) A fearless and faithful environment can be established.

3. ***Principle 3 - Involvement of People:*** People, at all levels, are valuable resources of an organization, and their full involvement enables their abilities to be used for the organization's benefit. Every resource at an independent level is very vital for the organization. Their commitment to the fullest enables them to use it for the organization's benefit.

The advantages associated with this principle are:

(a) Inspired, committed, and responsible people within the organization.

(b) Innovation in giving out ideas within the organization's objectives.

(c) Employees motivated to participate in and contribute to continuous improvement and hence improving their own performance.

Applying this principle will lead to:

(a) An opportunity for employees to enhance their competence, knowledge, and experience.

(b) Sharing of knowledge and experience, openly discussing problems and issues.

(c) More self-dependent employees who may take responsibility of problems and can solve those.

4. ***Principle 4 - Process Approach:*** Excellent results are achieved only when activities and related resources are managed as a process towards achieving the organization goal.

   The advantages associated with this principle are:

   (a) Perfect utilization of resources.

   (b) Enhanced, reliable, and expected outcome.

   (c) Focused and prioritized improvement opportunities.

   Applying this principle leads to:

   (a) Organized way of defining the activities necessary to arrive at a desired result and clear responsibility and accountability for managing key activities.

   (b) Distribution of resources, methods, and matters that will develop key activities of the organization.

5. ***Principle 5 - System Approach to Management:*** If interconnected processes are managed as a system, it will contribute to the organization's value and perfection in achieving the desired goals.

   The advantages associated with this principle are:

   (a) Integration and arrangement of the processes, which, in turn, leads to achieving the desired results.

   (b) Enhanced confidence level of interested parties, which results in consistency in the products, effectiveness in managing and the efficiency of the organization.

   Applying this principle leads to:

   (a) Improved system structure and approaches to achieve the organization's objectives in the most effective and efficient way.

   (b) Regular and methodical measurement and evaluation of the system against enhancement.

6. ***Principle 6 - Continuous Improvement:*** The aim of every organization is to continuously improve the overall performance and bring in a good name to the organization.

   The advantages associated with this principle:

   (a) Better performance.

   (b) Suggestion of improvement actions at all levels to an organization's strategic intent.

   Applying this principle leads to:

   (a) Following a reliable organization-wide approach to continuous improvement of the organization's performance.

   (b) Setting goals to guide and measures to track constant development.

7. ***Principle 7 - Factual Approach to Decision Making:*** Efficient decisions can be taken with a proper analysis of data and information.

   The advantages associated with this principle are:

   (a) Up to date decisions.

   (b) Additional opportunities to demonstrate the effectiveness of past decisions through reference to factual records.

   (c) Enhanced opportunities to review, challenge and change views and conclusions.

   Applying this principle leads to:

   (a) Reliable and accurate information. This makes data accessible to those who want it.

   (b) Assessing data and making decisions based on factual analysis, balanced with experience and intuition.

8. ***Principle 8 - Mutually Beneficial Supplier Relationships:*** An organization and its suppliers are interconnected and a mutually favorable relationship enhances the ability of both to create value for both parties.

   The advantages associated with this principle are:

   (a) Creating value for both parties.

   (b) Instant reactions to changing market or customer requirements and expectations.

   Applying this principle leads to:

   (a) Establishment of fair relationships between short-term and long-term concerns.

   (b) Establishment of combined development and enhancement activities.

   (c) Inspiring, encouraging, and recognizing improvements and achievements by suppliers.

Organizations can adapt quality management procedures in various ways. Implementation of these principles depends on the kind of organization and the specific challenges it faces. It is beneficial for organizations to set up quality management systems based on these procedures.


Caution

It is important to follow the ISO principles in the order they are listed.


Task

Visit the ISO website and try to analyze how these principles are used in various organizations.

14.3 Software Engineering Standards

Software engineering standards provide information about development, testing and maintenance of particular software as per certain specific set of rules. Several associations such as International Standards Organizations, National Standards Institutes, and several other professional and industry organizations are engaged in developing, adapting, and enforcing Software Quality Assurance (SQA) project process standards.


Did you know?

A 1997 survey quoted by Moore (1999) lists 315 software standards developed by 46 different organizations.

There is an increasing trend among associations to discard local standards and follow international standards. The Institute of Electrical and Electronics Engineers (IEEE) and the associated IEEE Computer Society are among some of the organizations who are leading this trend.

One of IEEE's prime contributions lies in the creation and promotion of standardization. A subgroup of the IEEE standards operational group was started in 1976 to introduce and promote SQA standards, which was later published as the IEEE Software Engineering Standard Collection (in frequently updated editions). Standards, among which most of them can be classified as project process standards, turn out to be a key source for international standards.

The intentions of IEEE/EIA Std 12207, as resolute by the IEEE and EIA, can thus be stated as:

1. To set up an international standard model of universal software life cycle processes that can be recognized by the software industries across the world.
2. To develop understanding among business groups by application of universally recognized processes, activities and tasks.

The SPICE (Software Process Improvement for Capability Determination) project and the ISO/IEC 15504 software process assessment standard are combined initiatives taken by ISO and IEC The SPICE (Software Process Improvement for Capability Determination) Project was introduced in 1993 to gain control over this problem by establishing a standard software process assessment methodology.

SPICE is a universal proposal for maintaining the improvement of international standards for Software Process Assessment. There are three prime objectives related to these:

1. To set up a working frame as a standard for software process assessment.
2. To perform organizational assessments for the up-and-coming standards.
3. To support the technology transfer of software process assessment into the software industries across the world.

At present, the spice standards are the guidelines and the prime target of any software project being carried out. The ISO/IEC 15504 consists of 29 processes that the organization has to execute effectively to attain capability level 5 of CMM.

The processes are grouped into five subject areas. This is shown in table 14.2.

Table 14.2 Grouping Based on Subject Areas

Subject area	No. of processes
Customer–supplier (CUS)	5
Engineering (ENG)	7
Support (SUP)	8
Management (MAN)	4
Organization (ORG)	5

The umbrella standard ISO/IEC 12207A that is also called as software lifecycle process standard is among the most extensively used standards. Moreover, it is a standard for every customer and process in the SESC collection.

The international standard sets up a universal framework for software all through its life cycle starting from the beginning throughout its journey, and it attends the organizational framework of those software processes both from the system's technical point of view and from the enterprise's business point of view.

The standard is broadly considered as a base for world trade in software services. The standard is already implemented by almost all the major countries of the world or is underway for the rest of them.

The ISO/IEC 12207 standard has improved and revised over past standards in comparable areas. Moreover, it is defined at the process level rather than at the procedure level. Rather than providing the level wise requirements characteristic of a process, it illustrates ongoing responsibilities that should be achieved and preserved during the entire life cycle of the process.

ISO/IEC 15504 consists of a reference model that includes both a process dimension and a capability dimension. The process dimension describes the processes having five process categories, which includes customer-supplier, engineering, supporting, management and organization. ISO/IEC 15504 defines a capability level for each process, which include the following:

1. (level-0): Incomplete Process
2. (level-1): Performed Process
3. (level-2): Managed Process
4. (level-3): Established Process
5. (level-4): Predictable Process
6. (level-5): Optimizing Process

The process attributes that mark the ISO/IEC 15504 standard includes:

1. Process Performance
2. Performance Management
3. Work Product Management
4. Process Definition
5. Process Deployment
6. Process Measurement
7. Process Control
8. Process Innovation
9. Process Optimization

ISO 15504 is mainly used while employing the perspective of process improvement and capability determination, which becomes relevant mainly while evaluating the capability of the supplier process.


Case Study

Success Attained using CMM

ABC Technologies is a universal software company having delivery bases in Baltimore and India. It provides offshore enterprise software solutions and services to clients in Canada and USA.

This company relied on CMM for developing software consumption prospective as well as building the trust and assurance with its clients. ABC Technologies Limited launched their project with an extensive CMM introduction training. With the help of CMM's verified process development methodology, the experts quickly recognized the issues and prepared a roadmap for further improvement of the process. Frequent learning sessions proved to be extremely effective in defining new processes and developing the existing ones. Several training and advisory sessions were arranged to ensure long-term sustainable process improvement. ABC Technologies started collecting the benefits of these efforts, almost immediately.

The experts created an integrated maturity model which provides the description of a mature, capable process. It identified the practices that required to be implemented for more effective as well as

Contd..

advanced practices. It also applies to practices associated with levels of development ranging from unrepeatable to an established, well-managed process. Typical paths were suggested through various practices for attaining higher levels of development, and therefore, improve an organization's processes. Experienced CMM staff members formed the core team who helped in the implementation.

Conventional modified processes to fit its business model enabled SMV to attain high levels of efficiency in a comparatively short span of time. Evaluation ability of its managers had greatly improved, and this brought higher level of visibility in its projects. Most important of all, the average number of appreciation emails from its clients had increased considerably.

Question

1. How did ABC Technologies' Maturity model Framework guide them in the organization?

14.4 Summary

- CMM is the widely used and preferred software method of evaluation.
- CMM defines five-level steps for process understanding.
- Key Process Areas (KPA) is a group of associated activities that strives to attain a set of goals when performed together. It also sets up a process capability at the maturity level.
- ISO 9000 is a written set of standards published by an international standard writing body.
- There are 20 crucial elements in ISO 9000 standard.
- Software engineering standards define how particular software can be developed, or tested and maintained according to certain specific set of rules.

14.5 Keywords

Audit: An official examination and verification of the organization's performance.

Business Process: A collection of related, structured activities or tasks that produce a specific service or product.

Software Engineering Institute (SEI): It is a federally funded research and development center. It maintains a list of organization assessed for CMM since 2006.

Software Process Assessment: Defines the methodology in which software has to be developed.

Strategy: A plan of action designed to achieve a particular goal.

14.6 Self Assessment

1. State whether the following statements are true or false:
   (a) A quality management system is a realistic and sensible method which promotes a methodical approach.
   (b) Software engineering standards define how hardware devices function.
   (c) CMM standards are not important to an organization.
   (d) A maturity model can be utilized as a model for evaluation.
   (e) CMM model is basically linked to the area of software development.
   (f) CMMI was brought out in 2002 with the idea of improving the usage of maturity models.
2. Fill in the blanks:
   (a) _____ is the widely used and preferred software method of evaluation.
   (b) There are _____ maturity levels in the CMM.
   (c) In the initial level of CMM _____ are not well structured.

(d) _____ is a written set of standards published by an international standard writing body.

(e) _____ is an international initiative for supporting the development of international standards for Software Process Assessment.

(f) There are _____ crucial elements in ISO 9000 standard.

3. Select the suitable choice for every question

(a) How many CMM levels are present till date?

(i) One (ii) Two (iii) Three (iv) Five

(b) What does SPICE stand for?

(i) Software Process Improvement for Capability Determination

(ii) Software Process Improvement for Capability Evaluation

(iii) Software Procedure Improvement for Capability Evaluation

(iv) Software Procedure Improvement for Capability Determination

(c) What does ISO stand for?

(i) International Organization for Standardization

(ii) International Software Organization

(iii) Indian Software Operations

(iv) Indian Standard Operations

(d) When was the spice project started?

(i) 1985 (ii) 1995 (iii) 2005 (iv) 1999

(e) What is the main quality that organizations look out for reaching their goals?

(i) Leadership qualities

(ii) Client interactions

(iii) Finance relations

(iv) Employee hiring structure

14.7 Review Questions

1. "CMM is the widely used and preferred software method of evaluation." Justify.
2. "A maturity model can be a combination of structured levels." Explain.
3. "Humphrey planned the process maturity framework to assist Application Development (AD) companies and to increase the quality of their AD methods in five stages." Explain the stages?
4. "CMM is an exclusive model which depicts an organization's growth and change." Substantiate.
5. "The objective of every organization is to strive for excellence." How is this achieved by adapting ISO 9000?
6. "ISO brings in good quality standards to an organization." Discuss.
7. "Quality management principles in ISO bring out the best in an organization." Justify.
8. "There are 20 crucial elements in ISO 9000 standard." Which element, in your opinion, is related to testing?
9. Assume that you are part of the product development team. If you are asked to develop procedures for rejected materials, which element would it be part of among the 20 elements?

10. "Software engineering standards help a company in improving the quality of work." Explain.
11. "SPICE software engineering standard brought about good quality when it was adopted." Discuss.
12. "There are three prime objectives related with SPICE." State the three objectives.
13. "The spice standards are now the guidelines and the major focus of any software project carried out." Explain.

Answers: Self Assessment

1. (a) True (b) False (c) False
   (d) True (e) True (f) True
2. (a) CMM (b) Five (c) Process
   (d) ISO (e) Spice (f) 20
3. (a) Five (b) Software Process Improvement for Capability Determination
   (c) International Organization for Standardization (d) 1993 (e) Leadership qualities

14.8 Further Readings


Books

Patton.R (2009). Software Testing, 2nd ed. Pearson Education.


Online link

www.iso.org/
www.iso.org/iso/catalogue_detail.htm?csnumber=38932
www.ieee.org/portal/innovate/products/standard/ieee_soft_eng.html