CONCEPT OF SOFTWARE TESTING

Surbhi Agarwal
Jayaprakash B

BOOKS ARCADE

Concept of Software Testing

Concept of Software Testing

Surbhi Agarwal
Jayaprakash B

BOOKS ARCADE
KRISHNA NAGAR, DELHI

Concept of Software Testing

Surbhi Agarwal
Jayaprakash B

BOOKS ARCADE

Regd. Office:
F-10/24, East Krishna Nagar, Near Vijay Chowk, Delhi-110051
Ph. No: +91-11-79669196, +91-9899073222
E-mail: info@booksarcade.co.in, booksarcade.pub@gmail.com
Website: www.booksarcade.co.in

Year of Publication 2023

International Standard Book Number-13: 978-93-90762-60-6

CONTENTS

CHAPTER 1

INTRODUCTION OF SOFTWARE TESTING

Ms. Surbhi Agarwal, Assistant Professor,
Department of Computer Science Engineering, School of Engineering and Technology, Jaipur National University, Jaipur, India
Email Id- surbhiagarwal2k19@jnujaipur.ac.in

Software Testing

Software testing is a process of checking software applications and products for bugs and errors to ensure their performance is efficient. Testing in software engineering is a fundamental process of creating reliable and usable software products. That's what makes it necessary in guiding effective software development. Understanding the different types of software testing is what the need for Quality Assurance in software development stems from.

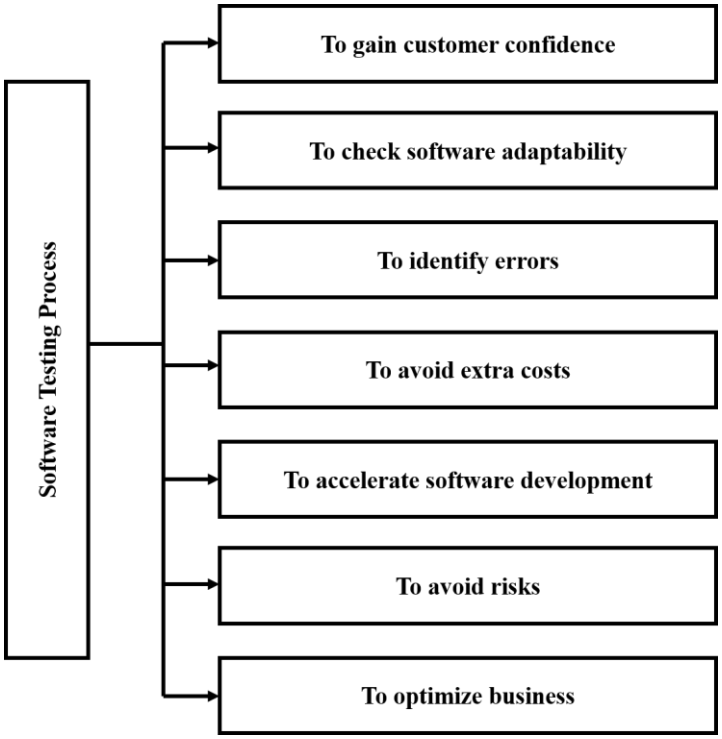


Figure 1.1: Represented that the Software Testing Processes.

Usually, companies that attempt to create software have dedicated software testers an in-house means of software testing help. By detecting errors and mistakes that affect the quality of software, these testers can ensure that software products are worthy of being sold in the market. Thus, they guarantee a software's usefulness, and help turning software applications into end-products that perform the way their designers intended them to.

Although the various types of software application testing can be more than a little long-drawn to fully appreciate, it's important for companies to be comfortable with why software testing is so essential in the first place. It's also important to note that the process of software testing is generally only a company's responsibility if they are software developers creating software for the market. If they're purchasing off-the-shelf software, especially one that's already been reviewed and

established, software testing has already taken place. In that case, it would make repeating the software testing process redundant.

They would also need to consider the purpose of testing if they're developing bespoke software for their specific needs. There can be various advantages for a company to invest resources in developing their own software, and that's something every business should evaluate on its own terms. For now, let's list the important reasons as to why software testing must be considered mandatory as mention in below figure.

1. To gain customer confidence
2. To check software adaptability
3. To identify errors
4. To avoid extra costs
5. To accelerate software development
6. To avoid risks
7. To optimize business

1. To Gain Customer Confidence

Software testing makes sure that the software is user-friendly. That makes it capable of being used by the customers it is intended for. Those who specialize in software application testing are familiar with the needs of customers, and unless a software can satisfy a customer's needs, it would be a practically useless investment.

Different kinds of software have different kinds of customers. For instance, a Human Resources department in a company that needs to track its employees and their performance would have an entirely different software package from a hospital administrator trying to evaluate which hospital departments need more resources. That's why just like software developers, software testers also tend to specialize in certain kinds of software designs. That's what makes software testing all the more resourceful in gaining customer confidence. It's a process that's case-sensitive, and takes care to make customers happy

2. To Check Software Adaptability

Given that there are many devices, operating systems, and browsers available today, it is necessary for software to be compatible with all platforms in order to offer users a smooth user experience. If the functionality of software is affected by the change of devices, it can count towards a negative user experience.

Testing eliminates such errors in the performance while adding to the compatibility and adaptability of the software. Of course, one can design software that's exclusively limited for use on a desktop, but given that so much of networking and work is now also conducted on smartphones, how useful would that software really be? Likewise, fewer customers will choose a software that only efficiently operates on an operating system such as that of the Apple Mac. It goes without saying that only the most adaptable software can really translate into success in the market.

3. To Identify Errors

While it seems intuitive, it's important to remember that software testing is what helps rid products of errors before the product goes into the hands of a client. Regardless of how competent software developers and engineers may be, the possibility of glitches and bugs is always present in untested software. Testing will lead to better functioning of the product as hidden errors will be exposed and fixed. Even a single bug can be damaging for the reputation of a software developing house. It can take a long time to regain customer confidence, so it's much better and ultimately more convenient to ensure testing is being achieved. The various types of software testing involved in the process ensure that the end result is software that will be valued by clientele.

4. To Avoid Extra Costs

Tied to the problem of errors is the issue of the costs of reimbursing clients who have experienced glitchy software. These additional expenses can amount to a significant amount of damages, as not only has the client been dissatisfied with the product for which they have paid, but a client's time that they could have invested elsewhere was also rendered worthless. That's why it's a misconception to believe software testing costs a lot of stress, and that testers themselves "break" software. Testers do feed large amounts of data to software applications, but that's what is necessary in software testing: if a software cannot support large data-loads, what purpose does it achieve for businesses or individual customers? If anything, software testing is the more cost-effective approach of handling software applications. Testing alleviates the need for a constant cycle of upgrades and fixes, as software testers identify bugs and errors before any such problems can arise.

5. To Accelerate Software Development

Some companies also neglect software testing as they have spent too much time on the software development process, and need to quickly deliver the product to the client. While this may be a problem with time-management in how a software development team is considering its work delegations, it's also an issue of conceptualizing software testing. Software testing and software development if run in parallel, can accelerate the software development process and make it more efficient. Staging the design process in a way that makes it certain that both software testing and software development are happening simultaneously takes care to avoid such pit-falls in software development.

6. To Avoid Risks

The worst thing about bugs and glitches is that it indicates a software is not secure. Especially when it comes to software that is meant for organizations, errors or loopholes can lead to vulnerability. This can lead to huge losses of information to competitor businesses, and can also lead to a lot of communication errors within an organization. Thus, besides just being about a software developer's reputation or the annoyance of encouraging bugs in an application's usability, bugs can also lead to privacy leaks and data gaps that can cost more than either of those things.

7. To Optimize Business

Testing allows the end-product to achieve a higher quality standard before being made live. It also adds to the company's brand image as well as its profitability through reduced support costs. Essentially, every software developer's goal is customer retention, and every customer's goal is finding a service that's reliable and worth their money. Providing effective software thus, allows a business to become entrenched in a software provider's reputation.

History of the Software Testing

We all know that before the development of automated software testing, all scripts were developed and executed manually. In 2022, the majority of software testing is carried out with the aid of QA automation tools.

i. Initiate the Software Testing Term

In order to ensure that a software program or commodity performs as intended, software testing consists of determining if the actual results matches the anticipated ones. Logically, this phrase didn't exist until sometime later than Charles Babbage's construction of the "Babbage machine," the first mechanical computer, in 1822. The history of system flaws and the planning of software testing to get rid of them actually started then, in the 19th century. Thomas Edison is credited both with the initial usage and subsequently invention of the word "bug" in 1878. Continued to work mostly with hardware, Edison sent a communication to a colleague in which he introduced the term bug. The correlation was great, and it immediately spread to other countries as little more than a phrase for software flaws. All of my creations have worked perfectly. Before commercial success or failure is inarguably reached, months of intense having to watch, study, and employment are required. The first step is really an intuition that comes with something like a burst, then difficulties arise the whole thing gives out and then that "Bugs" as such small faults and struggles are called show their self.

ii. The Father of Software Testing

People made significant advancements in computer technology through World War II, creating robust mechanical computers that could decipher encrypted messages and learn the procedures used by adversaries to correspond. The phrases "computer bug" and "debugging" were officially recognized in the history of unit testing by Grace Murray Hopper at the end of the Great War in 1945. Debugging is actually a very small portion of the whole assessment process, thus there are obviously many similarities between the two. Realizing this, professionals from all over the globe worked for a process that not only would eliminate problems but also ensured that the finished product behaved as planned. Software quality management was first addressed in 1951 by Joseph Juran, who is commonly considered as the founder of software testing, in his booklet Quality control handbook. Additionally, he had recognized the three components of QA management: planning, control, and improvement.

As project and program has progressed over the years, testing, a crucial component of software development, however has undergone a number of modifications. The foundation of it all was the programming and programming phases, when identifying flaws during debugging was seen as assessment.

Testing was given a distinctiveness and handled as a distinct profession from the debugging process in 1957. Testing was seen through till late 1970s as a process to make sure that the system met the requirements. After as well, in addition to making sure the technology was running properly, it was enhanced to detect the faults. The validation process was also thought of as a way to measure quality in the 1980s. As a result, it was allocated greater significance and then was handled as a process that was a component of the software development life cycle and was explicitly defined and monitored. The inspection process established its own life cycle by the middle of the 1990s. For a deeper understanding, we have divided the stages of software testing progression into distinct stages in this chapter.

iii. Era of Programmers and Testers in Software Testing

During this time, development and testing were seen as mutually independent tasks. The testing team received the software once it was finished and verified it. During the requirement analysis phase, testers were not very actively engaged and only sometimes interacted with business stakeholders. They were heavily reliant on information that was imparted to them through documentation created during design and development or learning from programmers who wrote the code. The testing team's use of limited testing strategies was a result of their lack of understanding of the needs and expectations of the customers. Based on their comprehension of the documentation, the testers would create a test plan and conduct ad-hoc testing of the software. It is clear that there were certain restrictions, and the testing was not exhaustive.

iv. Era of Exploration and Manual Testing

Test automation, exploratory testing, and other approaches became popular in the late 1990s. Thorough manual testing was done with the use of test design and test cases. By examining the program within the scope of testing charters, exploratory testing empowered users to test and break the software in their own unique ways. The software development process, due to its rapid and widespread expansion, demanded more sophisticated analytical techniques. The gradual and iterative method of agile testing contributed to the achievement of this objective. Duplicate tests that were able to be completed thanks to iterative testing.

Testing approaches have changed in recent years and through a large number of authors, a great deal of testing research has been filled with confusing and sometimes contradictory terminology. A complete taxonomy of testing terms is published from the International Software Testing Qualifications Board (ISTQB) at http://www.istqb.org/downloads/glossary.html. The language used this time has been modified to follow the ISTQB definitions, which have been modified to follow IEEE Computer Society standards (IEEE-1983). Here's a helpful explanation to get you started.

Error:

People make mistakes. Mistake is a suitable substitute. We refer to programming errors made by individuals as bugs. Errors often spread; a requirements error may become more apparent throughout design and much more so during code.

Fault:

A fault is a mistake that has happened. It would be more accurate to state that a defect is the manner of expression of a mistake, such as narrative prose, Unified Modeling Language diagrams, hierarchy charts, and source code. Fault may be compared favorably to defect and bug. It might be difficult to find faults. An error of omission occurs when a component that ought to be present in the representation is absent. We may talk of faults of commission and faults of omission, since this indicates a helpful improvement. When we insert inaccurate information into a representation, we commit an act of commission. When we forget to submit accurate information, we commit errors of omission. Faults of omission are more difficult to identify and correct than the other two categories.

Failure

A failure takes place when the code related to a problem runs. This definition has two subtleties: first, failures only occur in executable representations, which are often understood to be source

code or, more accurately, loaded object code; second, failures are only associated with errors of commission. How can we handle errors that result from omission faults? This may be taken a step further; what about defects that never occur or may not occur for a very long time? Reviews detect flaws, which helps to avert many failures; in fact, well-done reviews might uncover omission flaws.

Incident:

A failure may or may not be immediately obvious to the user when it happens (or customer or tester). An incident is a failure's symptom that notifies a user to the possibility of a failure.

Test

Obviously, mistakes, defects, failures, and mishaps are a problem in testing. A test is when software is put through its paces using test cases. Finding mistakes or displaying proper execution are the two separate objectives of a test.

Test Case

A test case is a representation of a program activity that has an identity. It also provides a list of predicted outputs and inputs.

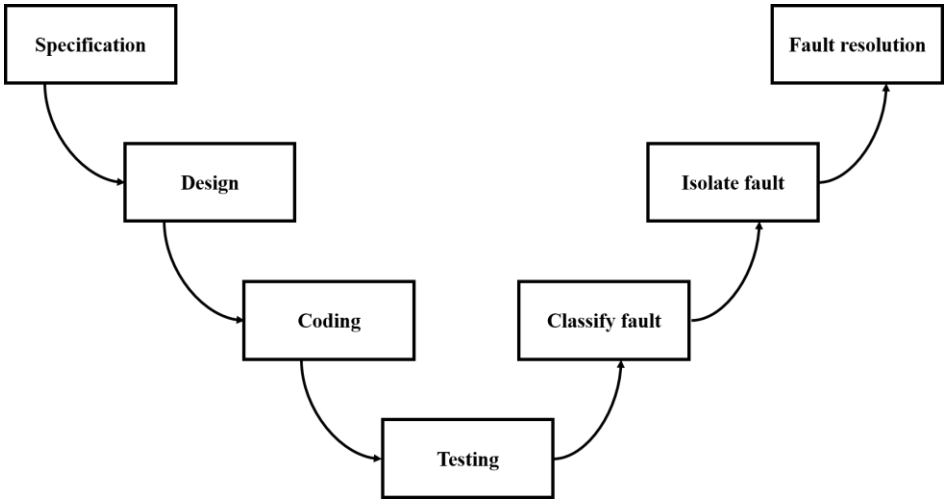


Figure 1.2: Represented that the Steps for Manual Testing

A testing life cycle model is shown in above figure and it is important to note that there are three primary methods for mistakes to be made all throughout development stages, which might lead to problems that could spread all throughout rest of the planning process. Another chance for mistakes and subsequent defects exists all through fault resolution process. A repair is flawed if it enables software that was previously acceptable act improperly. When we talk about test automation, we will go back to this. We can see that testing team play a key role in reviewing from the order of the phrases. Test planning, detailed test development, test case execution, and antibody test evaluation are all distinct components in the testing process. How to find based on the data collection of test cases is the primary concern of this book.

Importance of Software Testing

Software testing is an integral process in the software development life cycle wherein bugs, errors, and vulnerabilities are identified early on to ensure the security, reliability, and performance of a

software application. In addition to quality, software testing also contributes to the time-efficiency, and cost-effectiveness, and higher rates of customer satisfaction. Here's discussing 5-ways in which software testing helps companies write secure code, and enhance growth and productivity.

Decreased Software Development Costs:

Timely software testing eliminates the need of future investments in fixing issues that could have been avoided at an early stage. Even if errors or bugs do arise, it costs much less to resolve them. Therefore, software testing contributes to a cost-effective software development process.

Increased security:

As organizations are battling security risks, ingenious software testing methods are increasingly becoming the norm to provide trusted and reliable products. Software testing takes care of loopholes and entry-ways hackers can exploit to pursue malicious gains, thereby, averting potential security threats. It also ensures that personal information, banking details, and credentials are safe and secure.

Top-notch quality:

Software testing goes a long way in ensuring higher quality in an end-product. It ensures that there are no frequent crashes or bugs, and users have an uninterrupted experience. It is also carried out to determine the applications are providing top-notch functionality without causing glitches.

Higher rates of customer satisfaction:

Software testing is a guaranteed means to ensure customer satisfaction. With testing, you can discover the shortcomings of software, identify the problems that may impact customer experience, and improve them to contribute to customer satisfaction and retention.

High productivity and performance:

Companies that view software testing as an ongoing process and work with QA teams spend 22% less time on fixing overlooked issues.

This time is channeled towards completing value-adding work and developing innovative features that contribute to customer retention.

Helps in saving money

The testing of software has a wide array of benefits. The cost-effectiveness of the project happens to be one of the top reasons why companies go for software testing Services. The testing of software comprises of a bunch of projects. In case you find any bug in the early phases, fixing them costs a reduced amount of money.

Security

It is another crucial point why software testing should not be taken into consideration. It is considered to be the most vulnerable and sensitive part. There are a bunch of situations in which the information and details of the users are stolen and they are used for the benefits. It is considered to be the reason why people look for the well tested and reliable products.

As a specific product undergoes testing, the user can be ensured that they are going to receive a reliable product. The personal details of the user can be safe. Users can receive products that are free from vulnerability with the aid of software testing.

Quality of the product

For ensuring that the specific product comes to life, it should work in accordance with the following. Following the needs of the product is a prerequisite as it is helpful in getting the prerequisite results. Products should be serving the user in one way or the other. It is a must that it is going to bring the value, as per the promise.

Hence, it should function in a complete manner for ensuring an effective customer experience. It is also necessary to check the compatibility of the device. For instance, in case, you are planning to launch an application, it is a must to check the compatibility of the same in a wide array of operating systems and devices.

Satisfaction of the customer

The primary objective of the owner of the products is offering the best satisfaction of the customers. The reasons why it is necessary to opt for software testing is due to the fact that it offers the prerequisite and perfect user experience. As you opt for the best project in the saturated project, you will be capable of earning the reputation of reliable clients.

Thus, you are going to reap long-term benefits by opting for software testing. Earning the trust of the client is certainly not an easy task, primarily in case the product is found to be functioning and glitching every time or the other. You yourself have used a lot of products and you surely had several horrible experiences owing to which you might have deleted the application. The market is really saturated in the present days. The first impression is really important and if you fail to give the same, users are going to find another product which will accomplish all the requirements.

Enhancing the development process

With the aid of Quality Assurance, you can find a wide array of scenarios and errors, for the reproduction of the error. It is really simple and the developers need to fix the same in no time. In addition to this, software testers should be working with the development team parallelly, which is useful in the acceleration of the development procedure.

Easy while adding new features

The more interconnected and older the code, the more difficult it is to change. Tests counteract this calcification tendency by allowing developers to confidently add new features. As a new developer, changing older parts of your codebase can be terrifying, but with tests, you'll at least know if you've broken anything important. This helps in making your software stand ahead in the market, and beat the competition.

Determining the performance of the software

If you find software or application that has low or reduced performance, you will find that it brings your reputation down in the market. Users are not going to trust any people. There are chances that the reputation of your organization is going to suffer. In accordance with the experts, it is not that important. However, in case, you introduce any software in the market without software testing and after this, the performance of the software does not meet the expectation or requirements of the

clients, convincing people will be a hassle. Thus, software testing is considered to be an easy option as it helps in the determination of the performance of the software.

Different Software Testing Ideas:

Combining Automation & Manual Testing

This project highlights the importance of automation testing and manual testing to cover the security, performance, and usability aspects of software development. To ensure automation testing doesn't overlook user experience, and effectiveness of UI/UX design, it is important to incorporate manual testing in the software development process. In this manner, automation testers can handle the efficiency and speed aspects of QA while manual testers can test an application for its usability and user experience as mention in below figure.

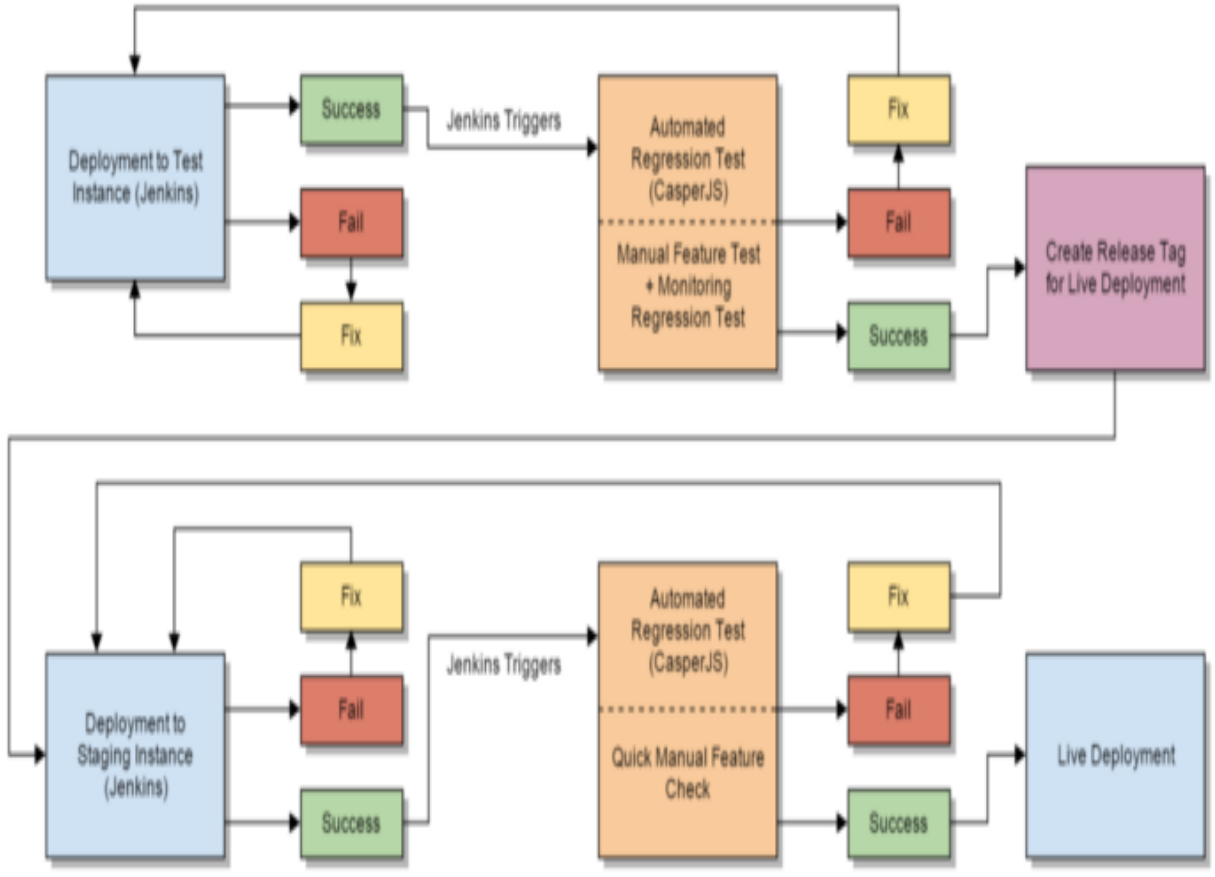


Figure 1.3: Represented that the various concepts of software testing

Testing Application Vulnerabilities Using Faulty Injection

This project employs a fault injector called "Pulad" to determine vulnerabilities in an application if any, prior to deployment. Pulad shifts from older approaches relying on static verification techniques that require executing the source code to reveal vulnerabilities. Fault injection, on the other hand, involves introducing bugs and errors to a system to determine its performance and

endurance. The process is carried out before the execution of the code, to ascertain how potent a system is to withstand potential faults, and recover from them.

Cross-Platform Tool to Build, Test and Package Software

CMake is an open-source family of tools hosted on GitHub and created by Kitware to provide a secure method to build, test, as well as package software. It allows developers to control compilation by generating native workspaces and makefiles. It is used with CDash which is a testing server designed to analyze, and view testing reports from anywhere around the world.

Software Testing to Combat Cybersecurity and Risk Compliance

With the digitization of business operations on the rise, 68% of business leaders report being wary of increasing cybersecurity risks. It is estimated that the worldwide information security market will reach $170.4 billion in 2022. This project highlights the necessity of software testing in protecting the privacy of end-users.

Software products and networks must benefit from secure coding practices to counter cyber-attacks and risk compliances. To do so, software professionals must invest in up skilling themselves to identify security threats and vulnerabilities and combat them.

Software Testing in IoT (Internet of Things)

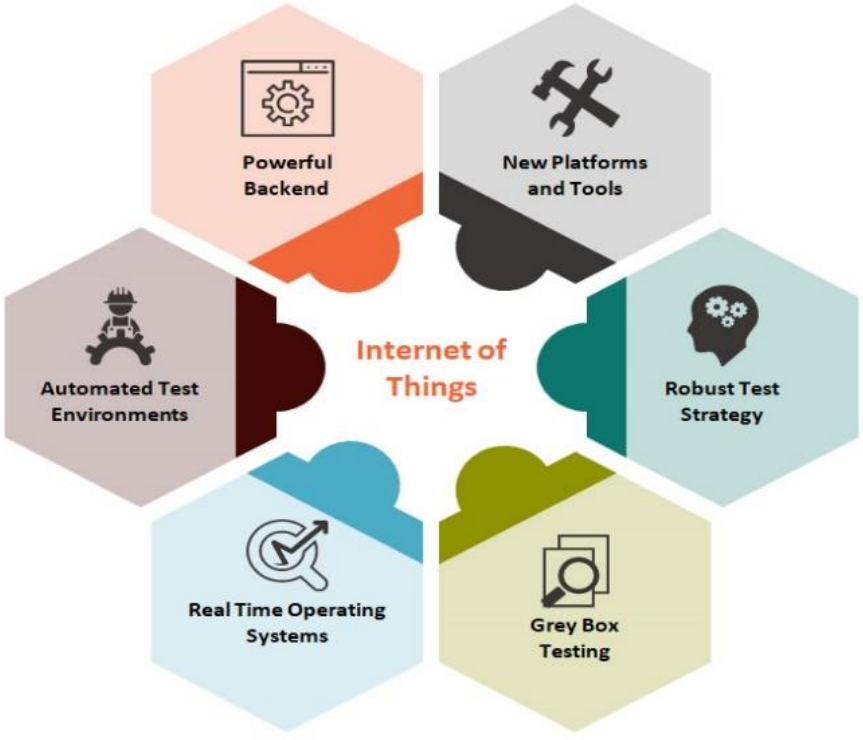


Figure 1.4: Software Testing in IoT (Internet of Things)

This project is to address the rise of the Internet of Things (IoT) technology-based devices that experience an estimated 5,200 attacks every month. As the global market of IoT is only going to progress from here (it is expected to reach US$1,102.6 billion by 2026), it is important for software testers to be aware of risks and security concerns IoT-based tools are likely to face in the future as

mention in below Figure. Software testers need to identify the usability and compatibility related risks to devise solutions to immediately mitigate risks. The thesis also addresses how until now a very small section of companies had been investing in Internet of Things testing strategies but the upcoming decades are projected to witness a rise in this sector.

Importance of Agile and DevOps Principles in Software Testing

Agile methodologies & DevOps are foundational principles of effective software testing around the world. The project focuses on using CI/CD principles to ensure rapid testing and deployment. Testing is carried out at different stages as developers verify the efficiency and performance of an application before releasing it into the real-world. Such practices in automated testing are proving to enhance the Quality Assurance process and resulting in better outcomes based on early bug detection, executing repeatable tasks, and benefits from constant feedback.

Automated Network Security Testing Tool

The project is based on Infection Monkey, an automated, open-source, security testing tool designed for reviewing a network's baseline security. It infects a system and allows users to monitor the progress of the infection and control it accordingly. It comprises multiple attacks, detection, and propagation capabilities.

Testing Angular Software

This project comprises software development tools, frameworks, and libraries to manage Angular projects. It is called Angular CLI and allows you to analyse and test Angular code, as well as create and manage it. Developers can use simple commands to generate necessary components and services, making running end-to-end unit tests easy and efficient.

Machine Learning and Artificial Intelligence to Enhance Automated Software Testing

It is no secret that AI usage will have a tremendous impact in almost every industry and aspect of creative technology.

It is estimated that the global market of Artificial Intelligence will be worth USD 733.7 billion by 2027. The aim of this project is to explore the role artificial intelligence and machine learning will play in software testing, especially in analysis and reports.

Some of the aspects of AI that are likely to impact automated testing are Test Suite Optimization, Log Analytics, and Predictive Analytics, among others. These are expected to help automated testers to determine the scope of additional testing required for an application and improve testing strategies through analytics and reports.

Different offers by Software Testing

With the world embracing more digital tools to create new efficiencies, businesses that want to stay relevant in the market need their offerings to remain dynamic, optimized for users, and pristine quality. Since it's often not possible to build, maintain, and enable technology alone, partnering with a software testing company is a great solution for continued success.

Getting into the capabilities and features of a prospective QA partner can help you move forward. Cutting through the advertising smokescreen of many technologies, this post will help you focus on the ten best features of leading software testing companies.

Manual Testing

There's no better tester than a human with hands-on experience. Manual testing involves real people performing runs on a mobile app or website, looking for bugs and performance issues. Manual testing usually results in writing test cases that include quality assurance (QA) assessments, reporting bugs or user experience problems as mention in the below figure. This feedback session is essential, and when someone with extensive knowledge performs it, you get detail-oriented results that are creative and well communicated. As a complement to automation, manual testing leverages more complex exploratory, usability, and localization testing. With manual testing, you can collaborate with another team to get the tests you need, how you need them.

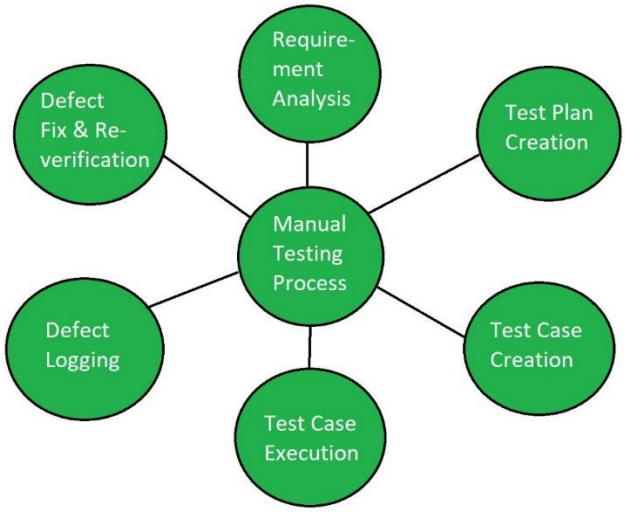


Figure 1.5: Different offers by Software Testing

Performance Assessment

This feature ensures that your software's performance is optimized. Companies who test performance often will include various smaller tests to ensure that everything is running as it should, such as:

- Load tests, which evaluate how the system works when handling multiple users.
- Stress tests, which evaluate how the system will behave when put under extreme loads.
- Spike testing, when you test short spurts of work on the system.
- Endurance tests, which will ensure that your software can handle extended work.
- Scalability, which checks if the system can be expanded.

Automation Testing

Automation and automated processes are ubiquitous in today's market, but maintaining automation over time can be incredibly time consuming. While companies shy away from automation because they lack the necessary resources to maintain and update the technology, a software testing partner that features automation testing can solve the issue.

When you have a suite of automated tests, it means that you have safety protocols in place that will wave down any issues when rolling out new updates. Ensuring that your automation processes work correctly can save your company hours by avoiding unnecessary work. In short, you're less likely to have to worry about whether your newest addition might cause a regression.

Real-Device Testing

When testing web and mobile apps, software partners should ensure dynamic offerings across operating systems, network connections, browsers, devices, and a global user base. Software development companies have two primary options for device testing real and virtual. Virtual device testing solutions like simulators or emulators are great for early-stage testing, while real device testing is crucial to ensure an app's success before it gets into the hands of end-users.

Often, utilizing a combination of testing options produces the best end product. Software testing partners that feature real device testing can cover a larger range of devices, users, and network realities.

Security Testing

With any web or mobile app, website, or online content, there's a risk that a third party will gain access to your servers and sensitive information if security systems aren't in place as display in Figure 1.6. By hiring a software testing company to test all vulnerable access points and block them, you can be sure your employees' and consumers' information is safe. Software testing partners as mention in below figure, that feature security testing often utilize the safest, most secure standards internally, which can help strengthen your trust in another company.

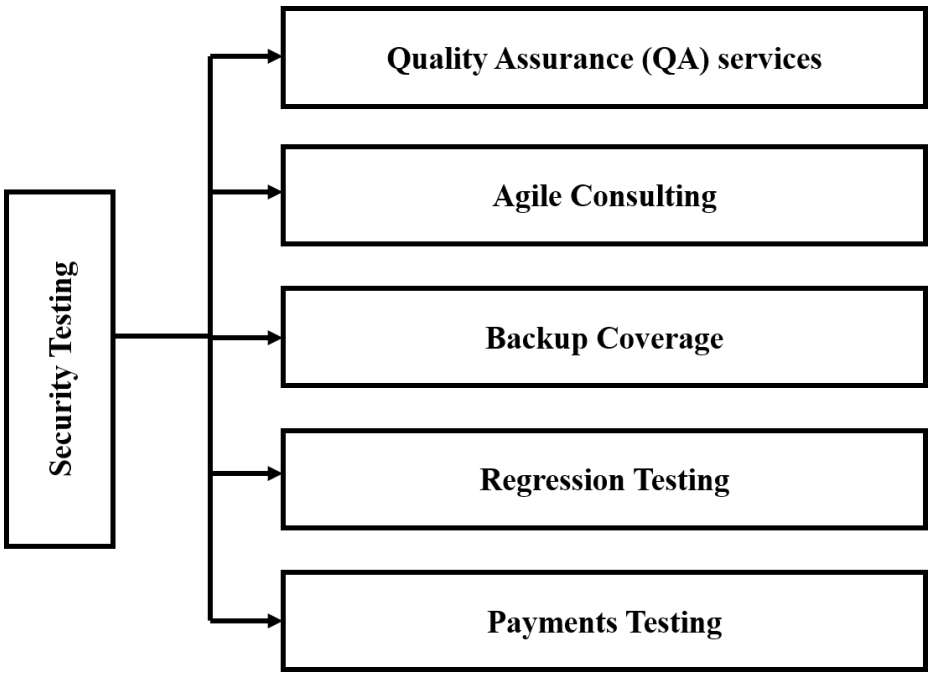


Figure 1.6: Represented that the Software Testing Process.

QA Services

Quality Assurance (QA) services help you understand that the product you're delivering is up to standard as well as giving you a concrete idea of your product's life cycle.

Agile Consulting

With agile consultants' and managed app testing services, you're offered an individual or team of experts who can set up, as well as audit, your testing processes. This means they will be able to start optimizing your workflow and testing, and in the software world, it is essential to keep evolving with requirements and times.

Backup Coverage

Partnering with a software company means adding additional staff to your repertoire, and the best partners ensure backup coverage. Runs, testing timelines, and release deadlines don't shift when freelancers take time off. Software companies should employ a global staff of freelancers and work teams to ensure quality checks are completed and run during holidays and time off.

Regression Testing

With any digital service or offering, eventually you will start to see wear and tear. As products, offerings, and websites are continually updating and iterated, software testing companies should offer a metric of secure regression testing. Having regression testing in place is an essential service that ensures freelance testers can log frequent defects, functionalities evident to users, and cases that verify the product's core features.

Payments Testing

For e-commerce enterprises, ensuring that your payment methods are secure and work properly is crucial for your company. While payments testing is a specific feature, leading software testing companies also integrate other types of testing to verify payment channels. Look for:

- Functionality testing
- Integration testing
- Performance testing
- Security testing
- Usability testing
- Compatibility testing
- Location testing

Opening up your QA strategy to include a software testing partner may feel like a risky maneuver, but limiting your product's growth, or worse yet, releasing a broken app to your customers, is even riskier.

Boundary Value Testing

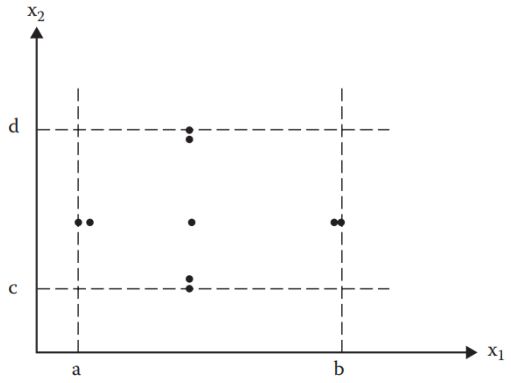
The domain and range of a functions may be the cross sections of other sets, and it transfers variables from one set to variables in another set. Any program may be thought of as a function inside this sense that its domain and range are controlled by its inputs and outputs. In this volume and the next two, we look at how to harness an understanding of a program's functionality to find unit testing for it. The most well-known specification-based form of assessment is input domain testing, often characterized as "boundary value testing." The responsibility is to develop has historically been the emphasis of this form of testing, but many of these technologies may be used

to create range-based test cases as a compliment. When assessing the input domain, there have been two separate factors to take into consideration. The first queries relates to whether we are worried about variables with incorrect values. Only valid input variable values are considered in standard boundary value testing. Robust boundary value testing needs to take into account both incorrect and correct variable values. The primary part is whether we adopt the dependability theory's standard "one fault" premise. This makes the assumption that errors result from an independent variable's wrong values. We must take the linear transformation of the selected variables if this is not justified and we are interested about the interaction between two or more of them. Together, the two variables result in four finite difference assessment variations:

- Normal boundary value testing,
- Robust boundary value testing,
- Worst-case boundary value testing,
- Robust worst-case boundary value testing.

Testing with Normal Boundary Value:

The border of the input space is the objective of all four types of boundary value testing in in order to identify test cases. Boundary value evaluation is justified by the discovery that mistakes often originate close to an input variable's extreme values. For instance, loop circumstances may test for something when they would test for something else, and counting often be "off by one." Using input value of a variable at their lowest, just above the minimum, a dollar value, just below their maximum, and at their maximum is the fundamental premise of boundary value management. Such test cases are produced for a suitably described software using a testing tool that is offered economically. Two well-liked front-end CASE tools have been successfully integrated with this tool. These numbers are referenced to by the tool as min, min+, nom, max-, and max. Two numbers, min- and max+, are added in the robustness forms. The "single fault" assumption, which is a crucial supposition in reliability theory, serves as the foundation of the subsequent component of boundary value analysis. This suggests that the simultaneous emergence of two or more defects seems to be very unlikely to cause failures. In contrast, the All Pairs testing method makes the observation that practically all errors in software-controlled healthcare treatments are caused by the interactions between two factors. By keeping all variables' values nominal, with the exception of one, and allowing that variable to take on all of its test values, neither normal and robust variations conditions are thus created. The different interventions for our function F of two variables under traditional boundary value analysis are represented in below figure.



Testing with Robust Boundary Value

A straightforward addition to standard boundary value testing is robust boundary value testing, which examines what transpires when the extrema are surpassed by variables that are just little above the maximum (max+) and just slightly below the lowest (min+) (min–). The majority of the topic of boundary value management, particularly the implications and restrictions, directly relates to sensitivity testing. It calculates the maximum from a robustness test are so much more interesting than the inputs actually. The aircraft might stall if it is the angle of attack of the wings. We hope nothing unusual happened if it's an elevator with a conventional load capacity. We would anticipate an error notification if the value was a date, such as May-32. Robustness testing's primary aspect is that it makes exceptional handling a priority. Testing for ruggedness in tightly typed programming may be quite challenging. When a variable in Pascal, for instance, is declared to fall within a specific range, values outside of that range lead to run-time errors that motivate students to crash. Robustness testing must be performed independently of the exception resolution option.

Testing with Worst-Case Boundary Value

As we said previously, the single fault assumption of weibull distribution is used in both types of boundary value testing. We discuss both standard worst-case perimeter testing and robust worst-case boundaries testing in this subsection due to their resemblance. By rejecting the single-fault hypothesis, we are intrigued in what transpires because many variables have extreme values. We employ the technique of "worst-case analysis," which is used in electrical circuit interpretation, to create worst-case test cases and test. We begin with the five-element set of the min, min+, nom, max-, and max values for each statistic. After that, we create test cases who use these sets. Insofar as boundary systematic evaluation test cases are an adequate subset of worst-case test cases, worst-case convex optimization testing is unquestionably more rigorous. Additionally, it necessitates considerably more work: compared to (4n + 1) test cases for boundaries value analysis, worst-case assessment for a function of unknown parameters creates 5n test cases. The generalization trend we saw for convex optimization analysis is followed in worst-case testing. The same regulations apply, especially those pertaining to independently. Worst-case testing is presumably most useful in situations when physical components interact heavily and failure of the function would have been obscenely expensive. We could use thorough worst-case testing for increasingly paranoid testing.

Testing for Special Value

The most popular such kind functional testing is undoubtedly special value testing. Additionally, it's also the least uniform and most logical. Special value research would analyze when a tester creates test cases integrating domain expertise, familiarity with communications components, and understanding of "soft spots." Another phrase for this would be ad hoc testing. Other than "best engineering judgments," no rules are employed. Special value testing hence heavily relies mostly on tester's skills. Despite all the major disadvantages, special value testing has its uses. You may discover test cases for three of our scenarios created using the strategies we just covered in the next section. You will notice that none of them, especially for the Next-Date function, are very suitable if you examine closely. Several test cases involving February 28, February 29, and leap years will be incorporated into the special value test cases for Next-Date. Special value screening is very subjective, yet despite this, it often delivers a collection of unit testing that are more

successful at ameliorates than those produced by convex optimization approaches, which is an testament to the skill of program development.

Life Cycle Based Testing

In this chapter, we start with a number of alternatives for the software development cycle and discuss how these frameworks affect testing. The waterfall method for software development has three layers, also including unit, integration, and system, which we identified from a broad perspective. This viewpoint has retained these levels for decades despite its limited effectiveness, but the entrance of other life cycle models compels a closer examination of these perspectives on testing. Since it is well-known and serves as a starting point for more previous research, we start with the conventional waterfall model. We next investigate variants on the waterfall paradigm before examining at some mainstream iterative development. We also undergo an important change in how we think. Because the representation potentially make it harder for us to detect test cases, we are more preoccupied with how to represent the product being tested. If you look at the papers given at the leading technology testing conferences, you'll see that there are almost as many speeches on specification frameworks and procedures as they are on testing methodologies. Software simulation and testing at all levels come together during model-based testing (MBT).

Traditional Waterfall Testing

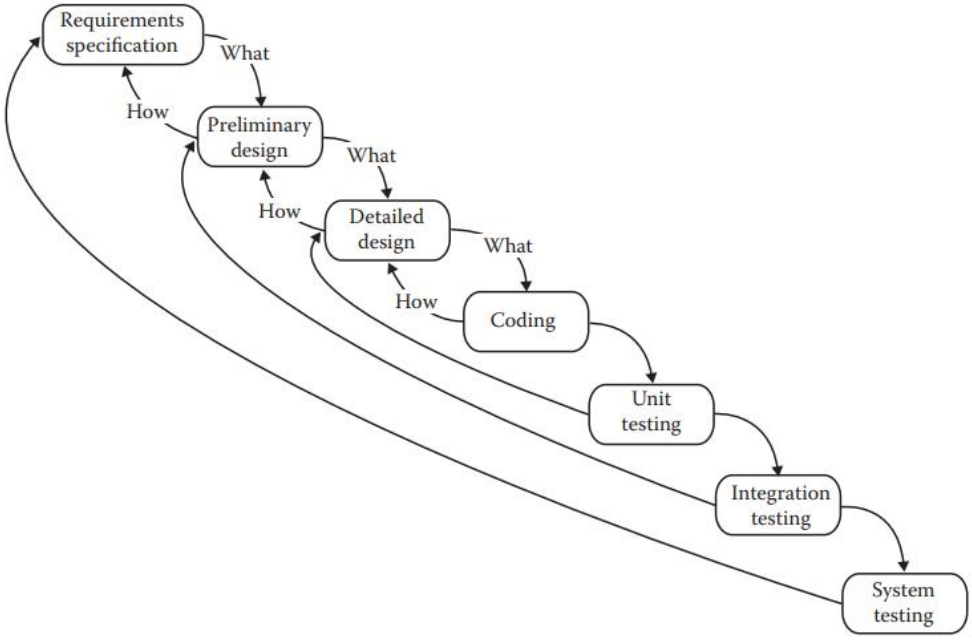


Figure 1.7: Represented that the Waterfall Testing Model.

In the below Figure 1.7, shows the waterfall paradigm, which has been employed traditionally in software development. To underscore how the fundamental levels of testing mirror these early waterfall stages, it is commonly shown as a "V," as seen in Figure 1.8. This viewpoint holds that perhaps the selection of test cases at that level is based on the data generated during one of the developmental phases. It's not controversial to say that we might anticipate that system test cases would be strongly connected to the requirements specification and that unit test cases would indeed be developed from the intricate unit design. The precise what-and-how cycles are essential on the waterfall's top left side. They underline how the predecessor phase establishes the prerequisites for

the succeeding phase. When accomplished, the succeeding's evident how it performs "what" was required. The happiest moments to examine software are also during these times. According to some talk show hosts, the stages on the left correspond to those that lead to the development of faults, while the ones upon that right lead to their discovery.

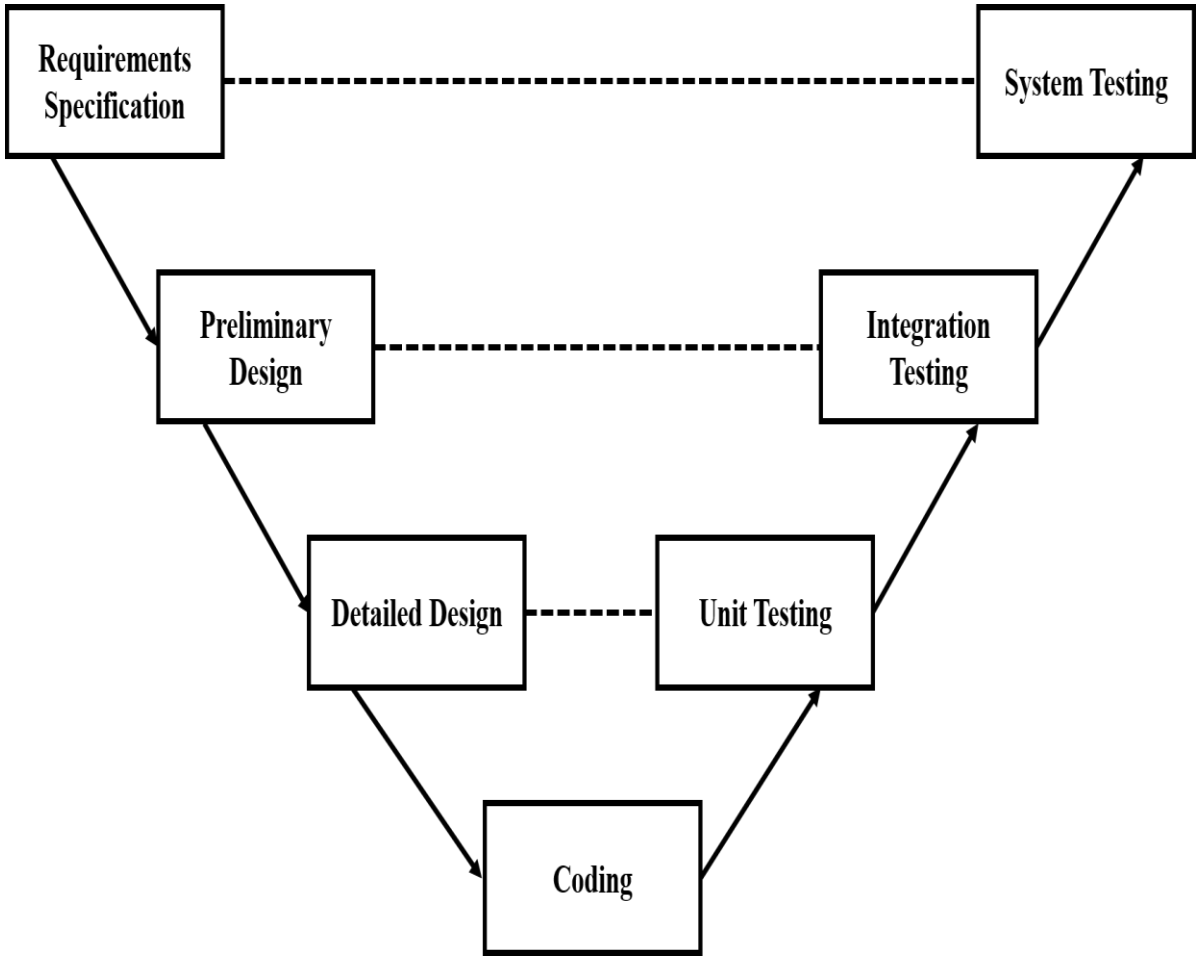


Figure 1.8: Represented that the V-shaped Model.

Waterfall Testing

The waterfall development method is intimately connected to top-down design and functional dissection. A divisional structure of the complete system into a tree-like structure of active constituents is the ultimate result of preliminary design. Top down integrations would start with the main program in this decomposition, examine the calls to the next level units, and so on until the decomposition tree's leaves was reached. Lower-level units are swapped out at each position with stubs, or temporary code, that performs the same functionality that the lower-level units would when called. The reverse order, defined as bottom-up integration, starts with the leaf units and progresses up forward towards the main program. Units at higher levels are replaced with motorists that mimic the procedure calls in bottom-up integration. The "big-bang" method just completes every unit at once, without any stubs or drivers. Traditional quality assurance aims to integrate previously tested modules with respect to the data flow diagram tree, regardless of the

technique used. Although integration testing generally described as a process, statements of this kind provide little understanding into the methodology or methodologies.

Pros and Cons of the Waterfall Model

Since its first publication in 1968, the waterfall-model has undergone a number of examinations and criticisms. The oldest concordance, which is a reliable source, records the following:

- The management structure's organization and the framework are compatible.
- The stages' finished product are well defined, thus makes project management easier.
- Individuals in charge of units may start work concurrently at the detailed design stage, reducing the time required for the project to progress overall.

More significantly, Agresti draws more attention to the waterfall model's significant flaws. We will discover that the developed life cycle models address these shortcomings. He notices that:

- The customer is not present throughout the lengthy feedback cycle that occurs between the defining of the requirements and the system testing.
- The model places a strong focus on analysis, almost to the expense of synthesis, which originates at the integration testing process.
- Due to manpower constraints, computationally intensive development at the unit level is unlikely to be viable.
- Most importantly, "perfect foresight" is important since any errors or omissions made at the specifications level would affect the succeeding life cycle stages.

The early waterfall designers were especially worried by the "omission" element. As an outcome, consistency, precision, and clarity were expected in almost every article on defining requirements. For the majority of needs definition approaches, consistency is difficult to establish but it is clear that clarity is required. The interesting characteristic is that all consecutive life cycles start out with discrepancy and rely on some kind of amplification to eventually hit "completeness".

Testing in Iterative Life Cycles

Practitioners have developed variations since the early 1980s in response to the inadequacies of the mainstream waterfall approach described in the previous section. The move away from implementation and toward a focus on iterations and composition is a feature recognized by all of these options. Decomposition nicely complements the waterfall model's top-down progression in addition to its bottom-up testing sequence, nonetheless it depends on one of the main weaknesses in waterfall development. "Perfect foresight" is required. Segmentation encourages analysis to the near exclusion that synthesis and can only be considered successful when the system is well understood. As a consequently, there is a very lengthy time between the defining requirements and the finished system, and throughout this time, there is no chance for user input. Contrarily, composition is more in line with how people genuinely work: it begins with a known as well as understood foundation, is subsequently expanded upon, and perhaps has unwanted elements omitted. Positive and negative sculptures may be compared using a really excellent analogy. As in the mathematician's understanding of Michelangelo's David: start with a chunk of concrete, and simply chip away but what that isn't David, negative artwork involves eliminating unnecessary material. Positive sculpture is often constructed using a malleable element, like wax. Wax is either

added or removed until the required form is created after approaching the middle shape. The wax replica is then plaster cast. The wax is melted away when the plaster solidifies, and the plaster "negative" is then utilized as a casting for molten bronze. Consider the implications of an error: with negative sculpting, so this whole piece must be destroyed and begun fresh. Positive sculpting only includes the removal and replacement of the defective component. We'll realize that this is the key characteristic of agile life cycle models. Integration testing is adversely affected by the alternative theories' emphasis on component.

Waterfall Spin-Offs

The waterfall paradigm has three principal derivatives: spiral mode, evolutionary development, and incremental development. As seen in below Figure 1.9, each of them includes a sequence of increases or builds. Instead of attempting to spread the cost such high-level design over a number of builds, it is crucial to maintain project proposal as an integral ingredient. When this is done, it often leads to undesirable design decisions being made in successive sessions as a result of early ones. The evolutionary as well as spiral models do not allow for this unique design stage. This is another considerable problem of bottom-up agile methodologies. The typical waterfall stages of thorough design through testing actually happen inside a build, with one crucial exception: testing procedure is split into two categories, regression and advancement testing. Debugging becomes essential as a result of the succession of builds. Retro testing's purpose is to ensure that features that were functional in an earlier iteration are still functional only with newest code. Regression testing would occur before, after, or even concurrent with system is significant.

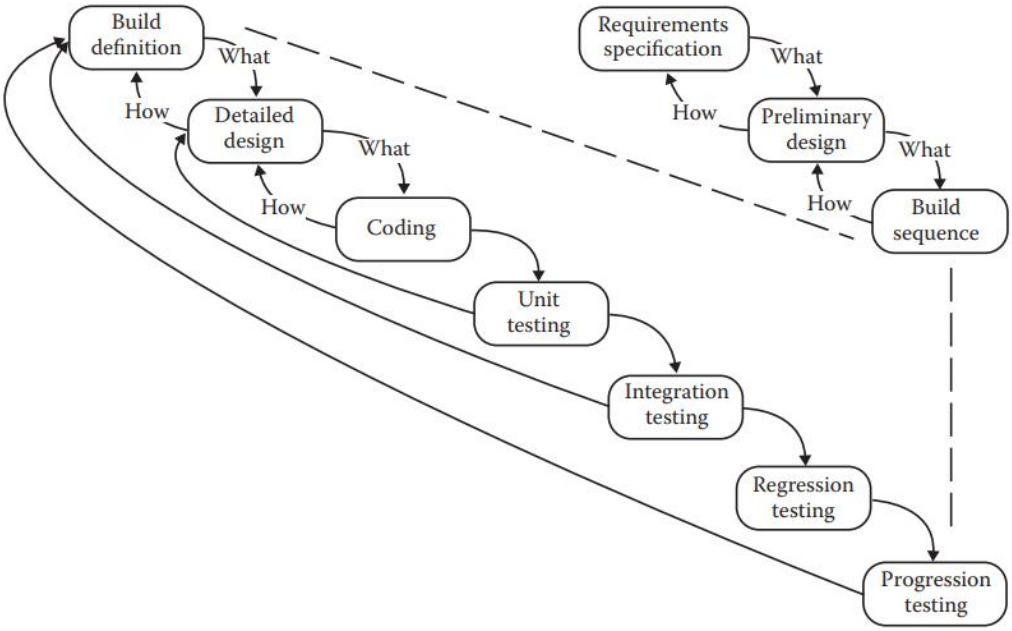


Figure 1.9: Represented that the Waterfall Spin-Offs Model.

Regression testing works on the concept that the new feature can be evaluated and that test automation was successful. We prefer to assume that adding new code demonstrates advancement rather than setback. A series of builds must absolutely include regression testing due to the well-known ripple impact of improvements to an existing system. Conforming to the industry standard, one improvement out of every five creates a new defect. The best way to describe genetic evolution is as client-based iteration. In this offshoot, customers are offered access to a limited first version

of a product before suggesting new entries. This is especially advantageous in cases where time-to-market is a top concern. A portion of the target market may be "locked in" to the first generation, making subsequent developmental versions more likely to resonate to that group. These clients are more likely to be engaged in the development of an application when they feel like they are "being heard" The spiral modeling approach by Barry Boehm incorporates elements of the creationist theory. The main differentiation is that risk, rather than customer ideas, is employed more frequently to set increments. The spiral is placed on an x-y coordinate plane, with the top left quadrant depicting setting goals, the upper right quadrant expressing risk analysis, the bottom right quadrant providing development, and the below left quadrant representing planning the very next iteration. In a providing practical, these four steps determining goals, changes in estimates, constructing and testing, and next iteration planning are repeated. The spiral develops larger with each developmental stage.

Specification-Based Life Cycle Models

Functional deconstruction is at best dangerous when systems are not completely understood by either the client or the developer. When Barry Boehm talks about the client who says, "I don't know what I want, but I'll recognize it when I see it," he makes fun of the situation. This is addressed by the fast prototyping life cycle, which is shown in below Figure 1.10, by supplying the "look and feel" of a system.

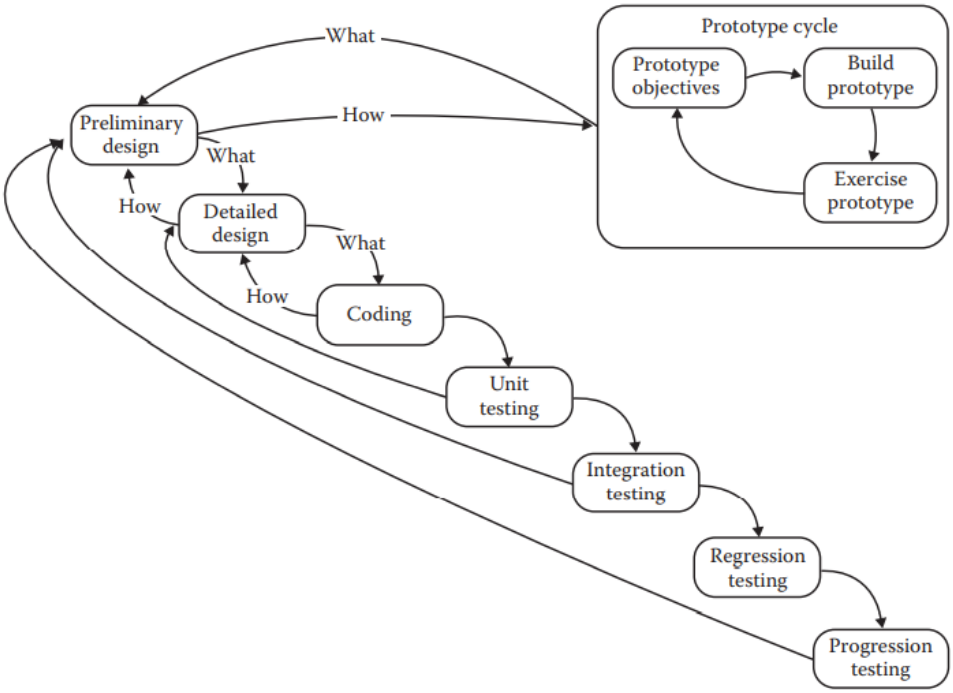


Figure 1.10; Represented that the Specification-Based Life Cycle Models.

Customers may so generously identify what they "see." This results in extremely early composition, which in turn subsequently reduces the specification-to-customer feedback cycle. A "fast and dirty" prototype is constructed rather than the complete system which is then utilized to gather consumer input. There may be subsequent prototype cycles as a result of the input. The development moves on and builds to the suitable specification after the client and developer believe that a prototype accurately depicts the required system. Any waterfall spin-off at this point

might also be utilized. The extreme of just this trend are the agile life cycles. Rapid prototyping offers some mind made implications for testing the system but no new consequences for integration testing. Where are the specifications? Is the most modern prototype the design? How are test cases for something like the system linked to the prototype? Using the prototype cycles as information-gathering operations before creating a specifications in a more conventional way is a useful response to concerns like these. Another option is to document whatever the client does with the prototypes, designate them as client-important scenarios, and deploy these as system test cases. These could be the predecessors of user stories from agile entire life cycle. Rapid prototyping's biggest benefit is that it adds the operations or behavioral perspective to the requirements conceptual design stage. Techniques for requirements definition often focus on a system's structure rather than its behavior. This is bad since the majority of individuals care more about behavior than structure.

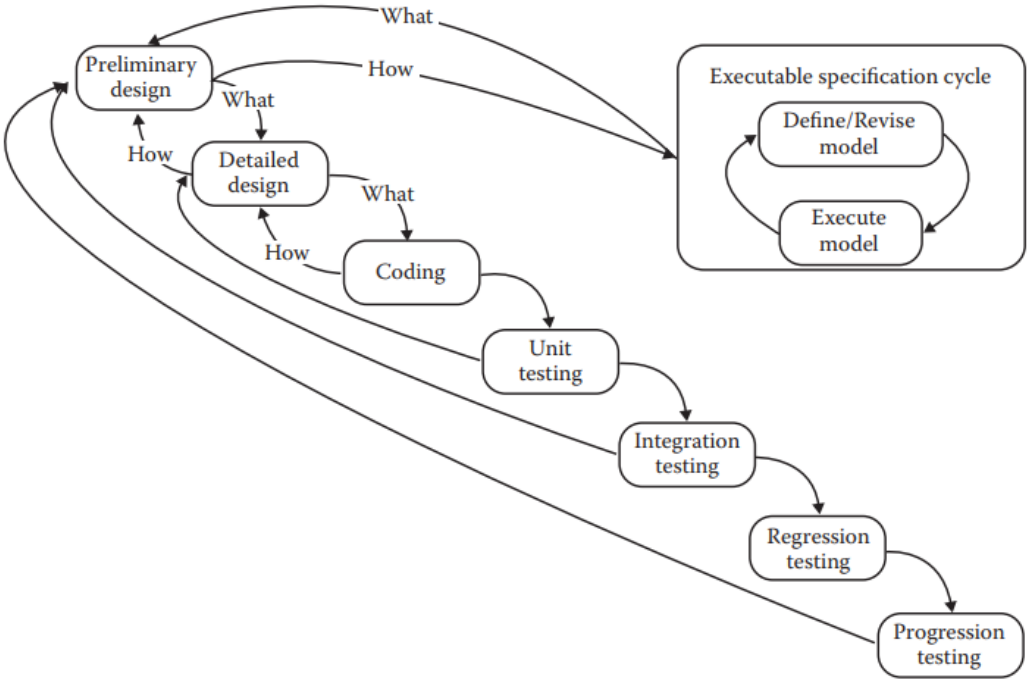


Figure 1.11: Represented that the Executable Specification Cycle.

The notion of quick prototyping is supplemented by the execution requirements shown in figure. This method specifies the requirements in a downloadable format as display in Figure 1.11. As in the fast prototyping methodology, the customer then puts the specification into operation to see how the expected actual performance behaves and gives feedback. The executable systems are, or have the potential to be, quite extensive. To say that State Charts is fully functional would be an overestimate. Expertise is needed to create an accessible model, and an engine is needed to implement it. Event-driven algorithms are ideally suited for the application of executable specification, especially when the events might occur in numerous sequences. Such platforms are referred to as "reactive" systems by David Harel, the developer of State Charts, since they respond to any outside events. An executable specifications serves the same objective as 3d printing in that it allows the client to encounter scenarios of desired behavior. Executable models may need to be changed in light of client input, which is another commonality. A competent executable model engine will facilitate the collection of "interesting" responsible to handle, and it is often a

practically mechanical approach to transform them into real system test cases. This is one unexpected advantage. System testing may be tracked back to the requirements if this is done correctly.

Again, there are no repercussions for integration testing from this life cycle. One distinguishing characteristic between a prototype and a system requirements document is that the latter is clear. More importantly, creating system test cases from that of an executable specification is often a mechanical process. Using an executable specification as the foundation for system testing creates an intriguing kind of system testing at the system level, which is another big distinction. The executable definition component may be integrated with any of the repetitive life cycle models, as learned from fast prototyping.

Dependability and Security

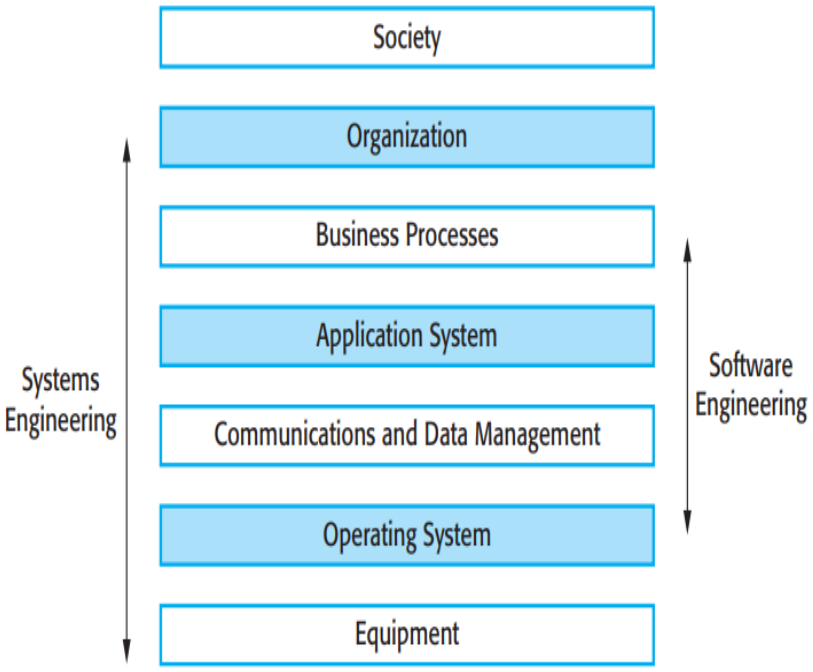


Figure 1.12: Illustrates that the Dependability and Security

Hardware and software work together to create a computer system. A software system is an approximation without hardware because it is merely a representation with certain human concepts and understandings. Hardware is a collection of inert computer devices without software. But if you combine them to assemble a system, you can build a technology that can do intricate calculations and broadcast the findings to its surroundings. The above exemplifies one of a system's core qualities that it is more than the sum of its parts. Systems have properties that are only visible whenever all of its parts are working as a single unit. Because of this, software engineering is not a standalone profession but rather a significant aspect of more comprehensive systems engineering techniques. Instead of existing in solitary, software systems are crucial components of larger systems to support a human, societal, or organization goal. For instance, the weather channel's instruments are managed by the software for the wilderness weather system. It interacts with other software applications and is a component of larger regional, national, and global climate modeling systems. These systems include of hardware, software, weather

forecasting techniques, system operators, and investigators of the system's results. The companies that rely on the system to assist them in offering weather predictions to residents, the government, business, etc. are also included in the program. Sometimes these larger systems have been referred to as scientific disciplines. Along with technical aspects like computers, software, and other equipment, they also comprise clear and understandable factors like individuals, procedures, rules, etc. Because of their space complexity, sociotechnical systems are almost challenging to comprehend in their entirety. As shown in given below Figure 1.12, you must see them as layers respectively. The developmental systems stack is made up of several layers

i. Equipment Layer:

Hardware requirements make up this layer, some of which might include computers.

ii. Operating System Layer:

This residing in areas with the hardware and offers a group of shared resources for the system's greater software layers.

iii. Communications and Data Management Layer:

This layer expands the features of the operating systems and provides an interface that enables engagement with more custom functionality, including access to distant platforms or a system database, for illustration. Due to its position between the computer system and the application, it is frequently referred to here as middleware.

iv. Application Layer:

This layer offers the essential application-specific functionalities. This layer may include a wide assortment of software modules.

v. Business Process Layer:

The overall organization processes that use the software system are established and put it into practice at this level.

vi. Organizational Layer:

Stronger competitive processes are comprised of this layer, along with environmental regulation, guidelines, and requirements that should be adhered to while utilizing the system.

vii. Social Layer:

The societal rules and laws that dictate how the specified by the user are established however at stratum.

Unexpected system issues are often left to programmers to fix since software is naturally adaptable. Consider that a radar facility has been constructed such that the radar image ghosts. The radar can really be moved to a location with less contamination since it would be unfeasible, and as such the systems experts must find an alternative solution to the ghosting. They should eliminate the ghost appearances by improving the software's image-processing powers. This might cause the software to run more slowly than desirable. The issue could therefore be referred to as a "software failure," while in reality it is a malfunction in the system's whole construction process. Software developers commonly find themselves in a circumstance where they must increase features and functionality without raising hardware expenditures. Many so-called software failures really come from seeking

to adapt the technology in order to satisfy altered requirements engineering requirements rather than due to inherent software vulnerabilities. A good illustration example of this was the luggage technical failure at the Denver airport, where the management software was supposed to accommodate for the equipment's limits. The process of creating complete systems, not simply the software underlying them, is known as computer engineering. Software engineering expenses often represent the largest cost component of the total system costs since it is the coordinating and integrating portion of these systems. A deeper understanding of how software works with these other equipment and programming and how it is intended to somehow be utilized is beneficial for software developers. This knowledge will assist you to engage in a systems technical department, comprehend the boundaries of software, and create good software.

Typically hierarchical, complex things include additional systems. An incident's location may be described using a geographic information system, for instance, in a police system of command and control. These integrated systems are often referred to as "subsystems." Subsystems have the capability to operate as separate, driverless cars. The same geographic information system, for examples, may be utilized in emergency commanding and control and transport warehousing facilities. Systems that include software may be divided into two categories

i. Technical Computer Based Systems:

These systems consist including both software and hardware parts, but lack operations and procedures. Televisions, mobile phones, and some other devices with application systems are examples of contemporary systems. This category also includes most PC applications, video games, etc. Technical systems are implemented by people and organizations for certain purposes, but the systems generally are unaware of this purpose. For examples, the word processor I'm employing isn't aware that I'm composing a book in it.

ii. Sociotechnical Systems:

These include one maybe more technological systems, but perhaps more importantly, they also incorporate individuals who are aware of something like the system's functionality inside the system itself. Peoples are integral aspects of scientific disciplines, which have specified work methods. They may be impacted by any outside restrictions like national legislation and regulatory regulations in adding to being managed by company guidelines and procedures. For instance, this volume was produced using a sociotechnical production system, which incorporates a number of technical methods and tools.

Enterprise systems named sociotechnical systems are created to assist in achieving an organizational goal. This might be carried out to increase revenue, utilise less equipment in production, collect taxes, keep existing airspace secure, etc. Because they form part of an organizational environment, the office culture and the organization's procedures and guidelines have a bearing on the selection, development, and implementation of these systems. The platform's users are those who are touched by the organization's approach to management and their connections with others within and outside the company.

Understanding the organization culture in which sociotechnical technologies are employed is significant when attempting to develop them. If you don't, users and their administrators could reject the technology since it won't fulfil their company's expectations. A sociotechnical system's

specifications, design, and operation may very well be impacted by organizational characteristics from the health care context. These factors include:

i. Process changes:

The environment's procedures may need to be altered to accommodate the system. If so, schooling will undoubtedly be important. There is a chance that the users may oppose the application of the system if the changes sufficiently large or cause mass layoffs for some.

ii. Job Changes:

Users may become less knowledgeable or modify how they behave as a result of new systems. If so, users would vehemently oppose the system's implementation from the inside of the company. Designs that force supervisors to adapt their operational procedures to match a new technology are often disliked. The system may make corporate managers feel as if it is lowering their reputation inside the corporation.

iii. Organizational Changes:

The positive organizational power structure may alter as little more than a result of the system. People who control entrance to a sophisticated system, for instance, have a considerable amount of political clout if a sustainable development can be defined on it.

System Engineering

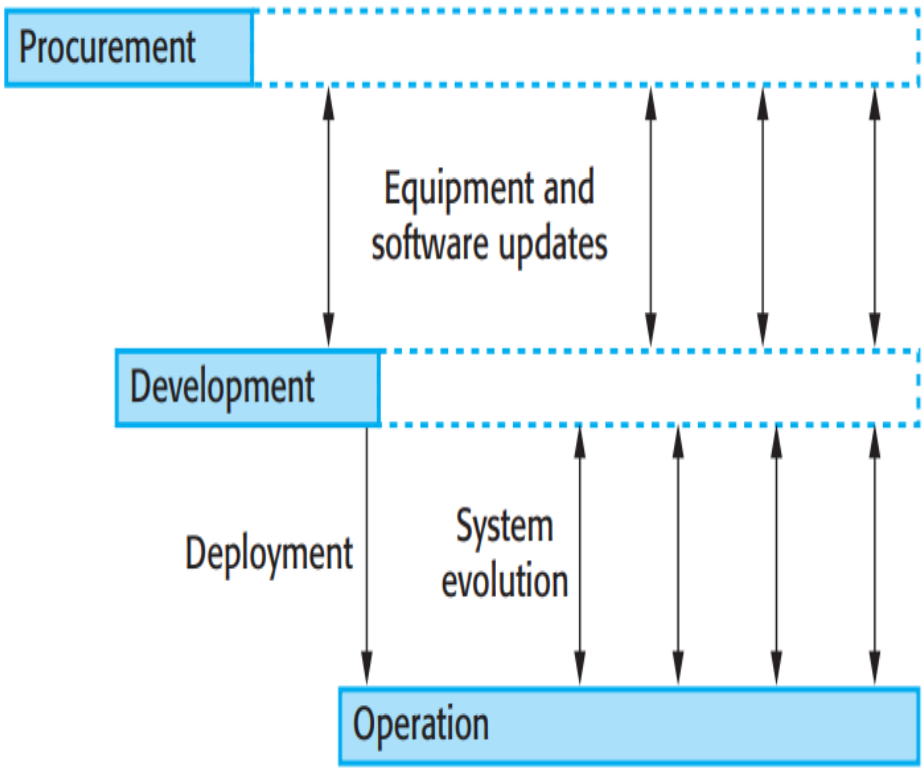


Figure 1.13: Represented that the Different System Engineering.

Systems engineering encompasses all of the activities involved in procuring, specifying, designing, implementing, and validating, deploying, operating, and maintaining sociotechnical systems. Systems engineers are not just concerned with software but also with hardware and the system's interactions with users and its environment.

They must think about the services that the system provides, the constraints under which the system must be built and operated, and the ways in which the system is used to fulfill its purpose or purposes.

There are three overlapping stages as mention in the given Figure 1.13, below, in the lifetime of large and complex sociotechnical systems:

i. **Procurement or Acquisition:**

   In this phase, a system's mission is chosen, high-level software and hardware are developed, choices are made about how functionality will be split between hardware, software, and employees, and the system's constituent units are bought.

ii. **Development:**

   The system is created at around this point. The activities involved in system development, such as objectives formulation, system-design, hardware and software-engineering, syste-integration, and testing, are all included in the development processes. The definition of standard operating procedures and the creation of system user training programs.

iii. **Operation:**

   At this point, the system is put into service after deployment and user training. The anticipated functional routines must then sometimes be modified to account for the platform's actual working environment. The system adapts over time when new needs are found.

   The system is eventually deactivated and changed when its value disintegrates.

They are not independent from each other. After the system is up and running, it could be needed to purchase new hardware and software to replace broken or antiquated system parts, to add functionality, or to handle a rise in demand. Similar to asks for adjustments during operation, software application is still needed.

System Procurement

System procurement, often also known as system acquisition, is the first stage in systems engineering.

At this stage, the decision is made on the system's scope, cost, and timeline, as well as the essential items for the system.

Further choices will be made based on this data on whether to purchase an equipment, the kind of system needed, and indeed the provider or suppliers of something like the system. These factors can affect these decisions:

i. **The state of Other Organizational Systems:**

   Purchasing a new system might have big impact on the organization's bottom line if it has a variety of challenging or suffer from high.

ii. **The Need to Comply with External Regulations:**

Investment in new solutions that increase business system performance may be prudent if a company has to compete more successfully or maintain its competitive position. A fundamental justification for purchasing additional military equipment is the need increase capacity in the midst of emerging threats.

iii. **External Competition:**

Restructuring is a normal place in businesses and corporations with the goal of enhancing performance and customer service. Business processes change as a result of reorganizations, necessitating new program support.

iv. **Business Reorganization:**

Businesses and other organizations frequently restructure with the intention of improving efficiency and/or customer service. Reorganizations lead to changes in business processes that require new systems support.

v. **Available Budget:**

The amount of new technology that may be purchased has been clearly influenced by the budget that is available.

Advanced Software Engineering

Reuse of Software

In reuse-based software engineering, the innovation process is focused on reusing already software application. Reuse as an agile methodology was first suggested more than 40 years ago, but the "development with reuse" started to become a standard for fresh business systems. Reuse-based development has grown more popular in response to calls for more affordable software creation and maintenance quicker system delivery, and better software. Software is quickly being seen as a significant asset by businesses. To boost their return on programming investments, they advocate reuse. Reusable software has become much more readily available. The open source revolution has made a large, relevant software base inexpensively accessible. This might occur in the form of complete apps or software libraries. There are multiple domain-specific application systems out there that may be customized and adjusted to meet the requirements of a specific business. A variety of modular components are offered by several major businesses towards their clients. It has become simpler to develop universal services and reuse services across a broad range of applications thanks to standards like web quality services. An method to development known as reuse-based computer programming aims to maximize the reuse of already developed software. The remanufactured software components might be of significantly varied sizes. For instance:

i. Application System Reuse:

It is possible to reuse an application system in its completeness by modifying it for other clients or by merging something without modification into other systems. As a replacement, application families with a collaboration between the two but customized for only certain clients might just be created.

ii. Component Reuse:

It is possible to reuse software components of all sizes, from subsystems to single objects. A pattern-matching number of technologies for a text-processing system, for examples, might be used to a database control system.

iii. Object and Function Reuse:

It is possible to reuse software requirements that carry out a single purpose, such as a mathematical theorem or an object class. For the last 40 years, standard standards have served as the basis for this kind of reuse. There are several free libraries of classes and interfaces. By connecting these libraries with freshly created existing applications, you may reuse the classes and functions provided inside. This is a particularly successful approach in fields like mathematics and graphics, where specialized knowledge is required to create efficient products and methods.

CHAPTER 2

AN INTRODUCTION TO SOFTWARE LIFE CYCLE

Ms. Surbhi Agarwal, Assistant Professor,
Department of Computer Science Engineering, School of Engineering and Technology, Jaipur National University, Jaipur, India
Email Id- surbhiagarwal2k19@jnujaipur.ac.in

Software Life Cycle

The software life cycle typically includes the following: requirements analysis, design, coding, testing, installation and maintenance. In between, there can be a requirement to provide Operations and support activities for the product.

i. Requirements Analysis:

Software organizations provide solutions to customer requirements by developing appropriate software that best suits their specifications. Thus, the life of software starts with origin of requirements. Very often, these requirements are vague, emergent and always subject to change.

Analysis is Performed:

To conduct in depth analysis of the proposed project, to evaluate for technical feasibility, to discover how to partition the system, to identify which areas of the requirements need to be elaborated from the customer, to identify the impact of changes to the requirements, to identify which requirements should be allocated to which components.

ii. Design and Specifications:

The outcome of requirements analysis is the requirements specification. Using this, the overall design for the intended software is developed.

Activities in this Phase:

Perform Architectural Design for the software, Design Database (If applicable), Design User Interfaces, Select or Develop Algorithms (If Applicable), Perform Detailed Design.

iii. Coding:

The development process tends to run iteratively through these phases rather than linearly; several models (spiral, waterfall etc.) have been proposed to describe this process.

Activities in this phase:

Create Test Data, Create Source, Generate Object Code, Create Operating Documentation, Plan Integration, and Perform Integration.

iv. Testing:

The process of using the developed system with the intent to find errors. Defects/flaws/bugs found at this stage will be sent back to the developer for a fix and have to be re-tested. This phase is iterative as long as the bugs are fixed to meet the requirements.

Activities in this Phase:

Plan Verification and Validation, Execute Verification and validation Tasks, Collect and Analyze Metric Data, Plan Testing, Develop Test Requirements, Execute.

v. Tests Installation:

The so developed and tested software will finally need to be installed at the client place. Careful planning has to be done to avoid problems to the user after installation is done.

Activities in this Phase:

Plan Installation, Distribution of Software, and Installation of Software, Accept Software in Operational Environment.

vi. Operation and Support:

Support activities are usually performed by the organization that developed the software. Both the parties usually decide on these activities before the system is developed.

Activities in this Phase:

Operate the System, Provide Technical Assistance and Consulting, Maintain Support Request Log.

vii. Maintenance:

The process does not stop once it is completely implemented and installed at user place; this phase undertakes development of new features, enhancements etc.

Activities in this Phase:

Reapplying Software Life Cycle.

Various Life Cycle Models

The way you approach a particular application for testing greatly depends on the life cycle model it follows. This is because, each life cycle model places emphasis on different aspects of the software i.e. certain models provide good scope and time for testing whereas some others don't. So, the number of test cases developed, features covered, time spent on each issue depends on the life cycle model the application follows. No matter what the life cycle model is, every application undergoes the same phases described above as its life cycle. Following Table 2.1, are a few software life cycle models, their advantages and disadvantages?

Table 2.1: Illustrated that the Different Model's Strength and Weakness.

Sr. No.	Prototyping Model	Spiral Model	Waterfall Model
	Strengths	**Strengths**	**Strengths**
1.	It is possible to establish specifications more specifically and early.	It encourages the earlier developmental phases of software to repurpose existing software.	Stresses accomplishing a phase before proceeding forward.
2.	It is possible for both customers and developers to convey expectations more thoroughly as well as clearly.	Enables the establishment of quality goals through development.	Early planning, increased customer, and design are stressed.

3.	It is possible to swiftly and effectively examine requirements and layout choices.	Helps lay the foundation for future software technology development.	Highlights assessment as a crucial point during the life cycle.
4.	Early detection of more constraints and system failures.	Early mistake and unwanted choices are prevented.	
	Weakness	**Weakness**	**Weakness**
1.	Need a prototyping tool and experience on how to use it, each of which are expenses for the design company.	Rapid Application Creation is required for this procedure or is most often linked to it, which is logistically quite challenging.	Requires early life cycle necessities to be captured and frozen.
2.	The production system might just be based upon that prototype.	Compared to the waterfall model, the process is more challenging to control and requires a profoundly different strategy (Waterfall model has management techniques like GANTT charts to assess).	Depends on separating technology from needs.
3.			Only testing phase does not provide feedback to any earlier stage.
4.			Not possible for certain businesses.
5.			Places more emphasis on outputs than on operations.

Software Testing Life Cycle

Software Testing Life Cycle consist of six (generic) phases, which in display in below Figure 2.1:

A. Planning,
B. Analysis,
C. Design,
D. Construction,
E. Testing Cycles,

F. Final Testing and Implementation,
G. Post Implementation.

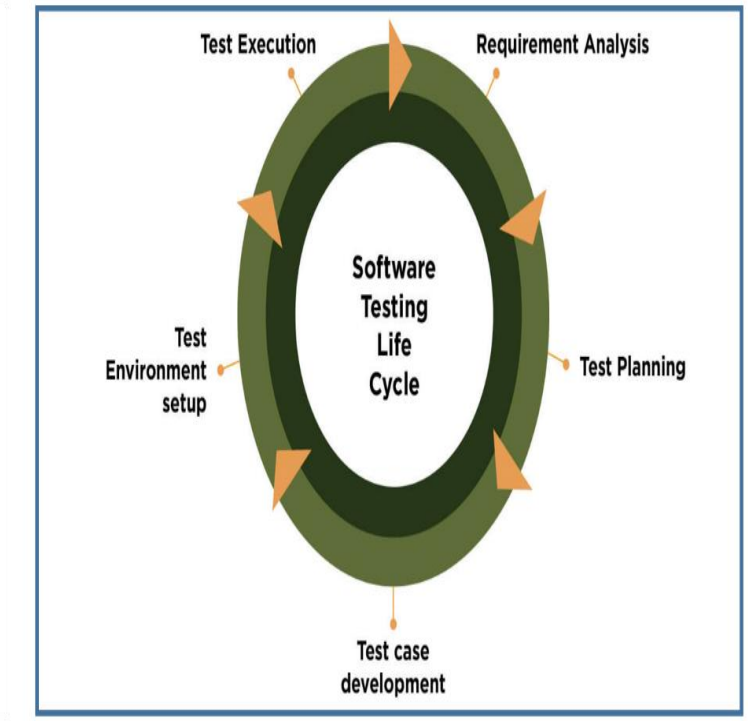


Figure 2.1: Represented that the Software Testing Life Cycle (STLC)

Each phase in the life cycle is described with the respective activities.

i. Planning:

High Level Test Plan, QA Plan (Quality-Goals), Identify Reporting Systems, Problem-Classification, Acceptance-Criteria, Testing D-atabases, Measurement Criteria (Defect Quantities/Severity Level and Defect Origin), Project Metrics, and lastly Start Project Testing Schedule. Plan to keep a repository of all test cases, regardless of whether they're manual or computerized.

ii. Analysis:

Involves developing functional validation depending on the business requirements writing test cases based on these details, developing test case format time estimates and priority assignments, developing test cycles matrices and timelines, identifying test cases to somehow be automated (if applicable), defining the area of mental anguish and performance testing, planning the different tests necessary for the project and integration tests, and defining data maintenance process (backup, restore, etc.).

iii. Design:

Verify that the test plan and scenarios are in a database and perhaps other required location, adjust test cycle matrices and timeframes, and other tasks throughout the concept stage. establish risk

management criteria, define specifics for stress and testing tools, complete test cycles (number of test cases per cycle according to time estimates per test case and priority), complete the test plan, keep posting test cases and add new ones in response to changes estimate resources to support development in unit testing.

iv. Construction (Unit Testing Phase):

All planning, test cycle patterns, schedules, manual test case passing touchdowns, stress and verification and validation, automated testing system testing, issue fixing to help development in unit testing, and QA acceptance test suite implementation must be completed before the technology can be handed over to QA.

v. Test Cycle(s) / Bug Fixes:

Run both the front-end and back-end test cases, report bugs, do verification, and update or add test cases as needed.

vi. Final Testing and Implementation:

Execute all Stress and Studies have reached, run all manual and computer controlled front end and back of the house test cases, provide ongoing defect tracking metrics, start providing ongoing complexity and design metrics, update test scenarios and test plan estimates, google docs spreadsheet test cycles, run test cases, and update as necessary.

vii. Post Implementation:

A meeting to assess the overall process after installation is possible. During this phase, the automating team will prepare the final Defect Report and related metrics, find solutions that will prevent similar issues in future projects, and thus more.

a. Examine test results and determine whether more instances should be automated for test automation,

b. Remove variables as well as test cases from automated tools,

c. A review of the methodology for combining the conclusions of manual testing with those derived from automated testing.

Definition of Bugs and Accruing Bugs

A coding mistake that culminates in an unanticipated deficiency, fault, flaw, or imperfection in a programming language is known as a software bug. In other words, there's most undoubtedly an issue if a software doesn't perform the way it should.

Software flaws may be attributed to ambiguous and very often functional specifications, software complexity, program flaws, schedule conflicts, tracking issues, ineffective communication, deviations from regulations, etc.

a. Program specifications that aren't explicit are a byproduct of misunderstandings as to what the software should and shouldn't accomplish. The intended use of the product might not be totally understood by the buyer.

b. This is especially relevant when software is created for something like a brand-new product. Such confrontations often result in many miscommunication on one or both sides.

c. The development and testing departments are under a lot of stress and uncertainty as a byproduct of the constantly changing software components. A new feature that has been introduced or an old feature that has been withdrawn from the programme often has connections to other modules or components. Bugs arise while such problems are ignored.

d. Additionally, if one issue in a software application is fixed, another bug may appear in the same or similar component. Inability to foresee such difficulties might result in major challenges and an increases in the number of bugs. Given that developers operate constantly under strict deadline, needs change frequently, there are more problems, etc., this represents one of the main reasons why bugs occur.

e. UI interfaces, module connection, database management systems, and developing and redoing them all all contribute to the complexity of the program and the system as a whole.

f. Fundamental issues in the architecture and design of software may result in programming issues. Software development errors are common since programmers are human. You may test for input/output errors, parameter errors, control flow flaws, data reference/declaration issues, and more.

g. The software may also be influenced by resource rescheduling, redoing or rejecting previously done tasks, and modifications to the hardware/software specifications. Bugs may arise if a new developer is brought on board mid-project. This is possible when relevant coding standards are not followed, when code specification is poor, when information sharing is inadequate, etc. If any old code is removed, it could leave a trace in other areas of the curriculum; ignoring or leaving this framework in place might result in issues. Larger projects seem to be more likely to have serious issues when it becomes more challenging to identify the source of the issue.

h. The closer the deadline, more and more hastily programmers work. During this stage, the overwhelming of bugs appear. You might well be able to identify bugs of multiple lengths and severity.

i. The difficulty of tracking the movement of all the problems may even result new bugs on its own. This becomes increasingly challenging if a bug has a highly complicated life cycle, meaning that it's been closed, reopened, rejected, forgotten, etc. more than once.

Software Testing Levels, Types, Terms and Definitions

i. Testing Levels and Types

Let will learn about many of the kinds of software testing in this chapter and how they may be used to that kind of Software Development Life Cycle. Software testing, though we all know, is the process of analyzing an application's functioning in compliance with the requirements of the client.

The many sorts of software testing must be conducted out if we want to be sure that their software is stable and free from bugs, considering testing is the only technique that can make an applications bug-free. Unit testing, integration testing, and system testing are the three fundamental testing tiers. These levels include a variety of testing types, as shown in below Figure 2.2.

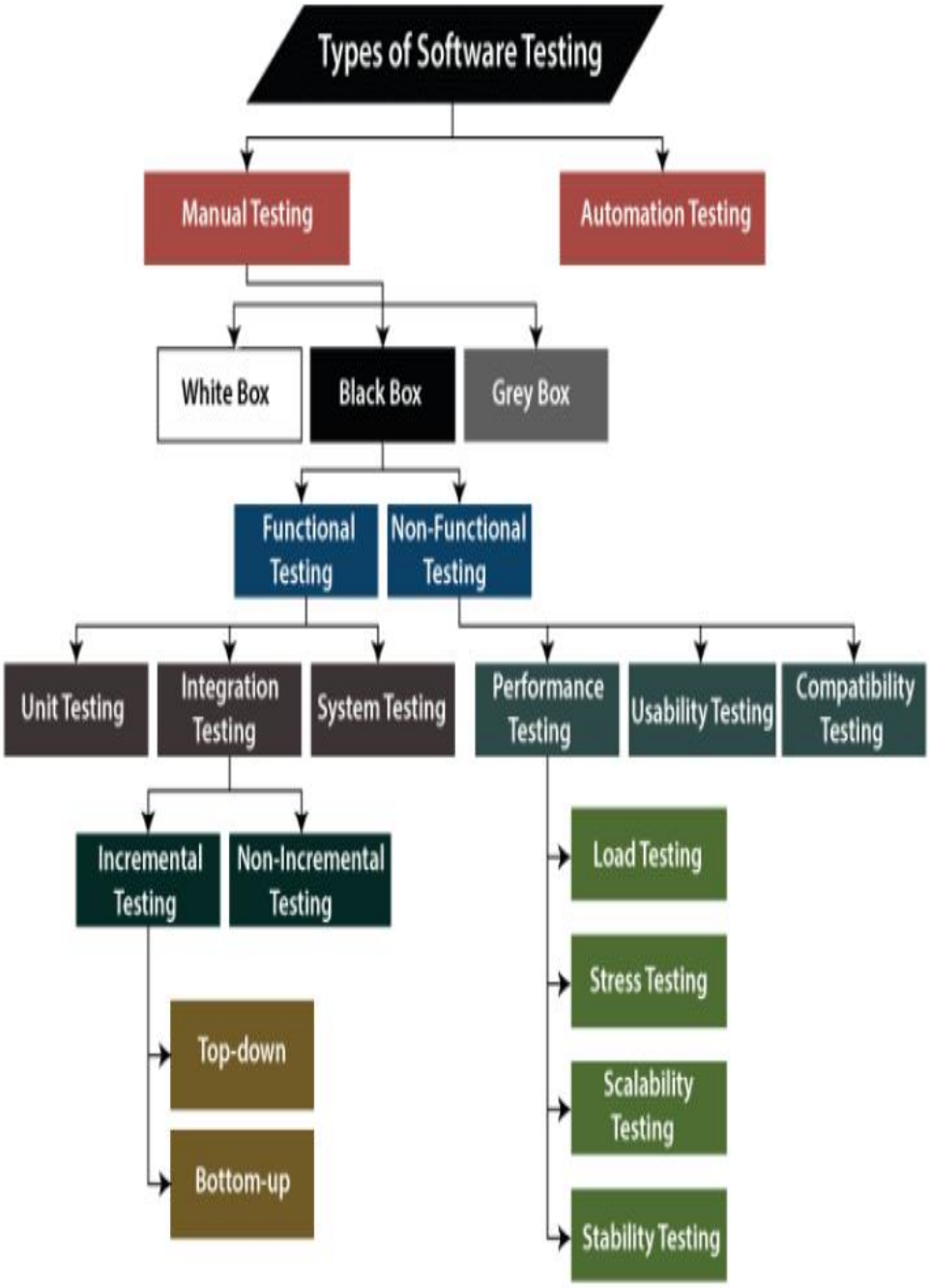


Figure 2.2: Represented that the Testing Levels and Types.

i. Unit Testing:

To examine a particular course or a component thereof.

ii. Integration Testing:

Unit testing has been carried out on each component that makes up a system to confirm interconnections between them.

iii. System Testing:

To compare the general system's actions to the platform's original aims. Software testing is an approach that determines if software software accurate, inclusive, and of high quality. The list of the several software testing categories and their description is shown below in a randomized order:

Formal Testing:

Performed by test engineers.

Informal Testing:

Performed by the developers.

Manual Testing:

That part of software testing that requires human input, analysis, or evaluation.

Automated Testing:

Software testing that utilizes a variety of tools to automate the testing process. Automated testing still requires a skilled quality assurance professional with knowledge of the automation tools and the software being tested to set up the test cases.

Black box Testing:

Software testing that utilizes a variety of tools to automate the testing process. Automated testing still requires a skilled quality assurance professional with knowledge of the automation tools and the software being tested to set up the test cases.

White box Testing:

Testing in which the software tester has knowledge of the back-end, structure and language of the software, or at least its purpose.

Unit Testing:

Unit testing is the process of testing a particular complied program, i.e., a window, a report, an interface, etc. independently as a stand-alone component/program. The types and degrees of unit tests can vary among modified and newly created programs. Unit testing is mostly performed by the programmers who are also responsible for the creation of the necessary unit test data.

Incremental Testing:

Incremental testing is partial testing of an incomplete product. The goal of incremental testing is to provide an early feedback to software developers.

Integration Testing:

Testing two or more modules or functions together with the intent of finding interface defects between the modules or functions.

System Integration Testing:

Testing of software components that have been distributed across multiple platforms e.g., client, web server, application server, and database server to produce failures caused by system integration defects i.e. defects involving distribution and back.

Functional Testing:

Verifying that a module functions as stated in the specification and establishing confidence that a program does what it is supposed to do.

End-to-end Testing:

Similar to system testing - testing a complete application in a situation that mimics real world use, such as interacting with a database, using network communication, or interacting with other hardware, application, or system.

Sanity Testing:

Sanity testing is performed whenever cursory testing is sufficient to prove the application is functioning according to specifications. This level of testing is a subset of regression testing. It normally includes testing basic GUI functionality to demonstrate connectivity to the database, application servers, printers, etc.

Regression Testing:

Testing with the intent of determining if bug fixes have been successful and have not created any new problems.

Acceptance Testing:

Testing the system with the intent of confirming readiness of the product and customer acceptance. Also known as User Acceptance Testing.

Adhoc Testing:

Testing without a formal test plan or outside of a test plan. With some projects this type of testing is carried out as an addition to formal testing. Sometimes, if testing occurs very late in the development cycle, this will be the only kind of testing that can be performed usually done by skilled testers. Sometimes ad hoc testing is referred to as exploratory testing.

Configuration Testing:

Testing to determine how well the product works with a broad range of hardware/peripheral equipment configurations as well as on different operating systems and software.

Load Testing:

Testing with the intent of determining how well the product handles competition for system resources. The competition may come in the form of network traffic, CPU utilization or memory allocation.

Stress Testing:

Testing done to evaluate the behavior when the system is pushed beyond the breaking point. The goal is to expose the weak links and to determine if the system manages to recover gracefully.

Performance Testing:

Testing with the intent of determining how efficiently a product handles a variety of events. Automated test tools geared specifically to test and fine-tune performance are used most often for this type of testing.

Usability Testing:

Usability testing is testing for 'user-friendliness'. A way to evaluate and measure how users interact with a software product or site. Tasks are given to users and observations are made.

Installation Testing:

Testing with the intent of determining if the product is compatible with a variety of platforms and how easily it installs.

Recovery/Error Testing:

Testing how well a system recovers from crashes, hardware failures, or other catastrophic problems.

Security Testing:

Testing of database and network software in order to keep company data and resources secure from mistaken/accidental users, hackers, and other malevolent attackers.

Penetration Testing:

Penetration testing is testing how well the system is protected against unauthorized internal or external access, or willful damage. This type of testing usually requires sophisticated testing techniques.

Compatibility Testing:

Testing used to determine whether other system software components such as browsers, utilities, and competing software will conflict with the software being tested.

Exploratory Testing:

Any testing in which the tester dynamically changes what they're doing for test execution, based on information they learn as they're executing their tests.

Comparison Testing:

Testing that compares software weaknesses and strengths to those of competitors' products.

Alpha Testing:

Testing after code is mostly complete or contains most of the functionality and prior to reaching customers. Sometimes a selected group of users are involved. More often this testing will be performed in-house or by an outside testing firm in close cooperation with the software engineering department.

Beta Testing:

Testing after the product is code complete. Betas are often widely distributed or even distributed to the public at large.

Gamma Testing:

Gamma testing is testing of software that has all the required features, but it did not go through all the in-house quality checks.

Mutation Testing:

A method to determine to test thoroughness by measuring the extent to which the test cases can discriminate the program from slight variants of the program.

Independent Verification and Validation:

The process of exercising software with the intent of ensuring that the software system meets its requirements and user expectations and doesn't fail in an unacceptable manner. The individual or group doing this work is not part of the group or organization that developed the software.

Pilot Testing:

Testing that involves the users just before actual release to ensure that users become familiar with the release contents and ultimately accept it. Typically involves many users, is conducted over a short period of time and is tightly controlled.

Parallel/Audit Testing:

Testing where the user reconciles the output of the new system to the output of the current system to verify the new system performs the operations correctly.

Glass Box/Open Box Testing:

Glass box testing is the same as white box testing. It is a testing approach that examines the application's program structure, and derives test cases from the application's program logic.

Closed Box Testing:

Closed box testing is same as black box testing. A type of testing that considers only the functionality of the application.

Bottom-up Testing:

Bottom-up testing is a technique for integration testing. A test engineer creates and uses test drivers for components that have not yet been developed, because, with bottom-up testing, low-level components are tested first. The objective of bottom-up testing is to call low-level components first, for testing purposes.

Smoke Testing:

A random test conducted before the delivery and after complete testing.

White Box Testing

Black box testing and white box testing are both aspects of the box testing method to software testing. In this chapter, we'll talk about white box testing, also referred to as test execution, clear box testing, accessible box testing, and transparency box testing. It examines a software's internal coding and infrastructures with an emphasis on correlating established inputs to anticipated and expected results. It is based on an application's business controls and focuses on structural and architectural testing. The design of test cases for such type of testing requires programming experience.

Focusing on the inputs or outputs via the application and enhancing its security are now the main objectives of white box testing. Because of the underlying system viewpoint, the expression "white box" is employed. The nicknames "clear box," "white box," and "transparent box" all refer to the accessibility of the software's intricacies. White box testing is conducted by developers. Every

lines of the program's code will be verified in this by the developer. The testing phase does the black box testing after that the developers have performed the white-box testing on the piece of software, at which point they deliver it back to the developer after discovering any issues.

The white box testing contains various tests, which is mention in below Figure 2.3:

1. Path testing
2. Loop testing
3. Condition testing

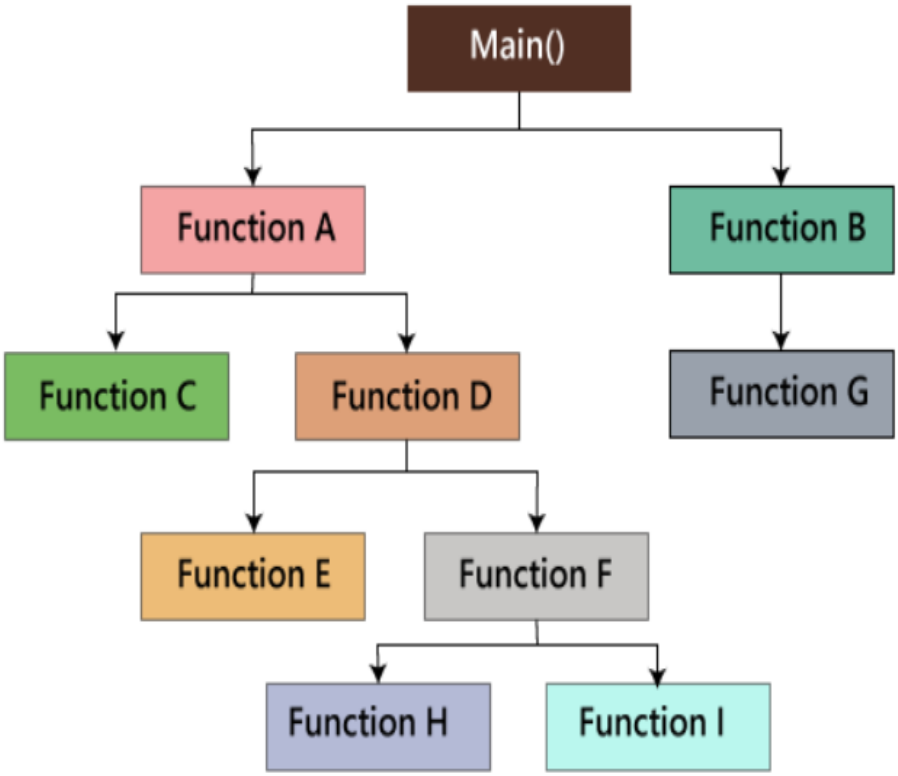


Figure 2.3: Represented that the various tests of White Box Testing.

i. Path Testing:

Users will create flow diagrams and test each autonomous route throughout the path assessment process. As we can see according to the above picture, writing the flow graph here suggests that flow graphics are expressing the flow of the program and also illustrating how every programed is combined with one another. And test all the autonomous pathways suggests that, in the situation of a path from main() to function G, the program should have been tested to see whether it is proper along that specific path prior to actually proceeding to test any other paths and repair any defects.

ii. Loop Testing:

While testing loops like while, for, and do-while, and some others, wit will also examine if the terminating condition is functioning correctly and where the size of the requirements is sufficient. For Example:

```
{
  while (5000)
   …….
   …….
}
```

These claims cannot independently test this software for all 5000 loop cycles. As shown in the program below, a little program that aids all 5000 cycles can be created. This program is characterized as a unit test and is only developed by development. Test P is written in a language that is analogous to that of the version control program.

```
Test P
{
--------
--------
}
```

As shown in the Figure 2.4 below, we have a number of needs, including 1, 2, 3, and 4. The developer then constructs programs like programs 1, 2, 3, and 4 for parallel conditions. The programs in consideration has hundreds of lines of code.

To locate the mistake, the developer will do white box testing, going line by line through each of the five applications. They will fix any bugs they find in any of the apps as mention in below figure. They then have to test the system once again, which takes a lot of time and effort and delays the period when the product is released.

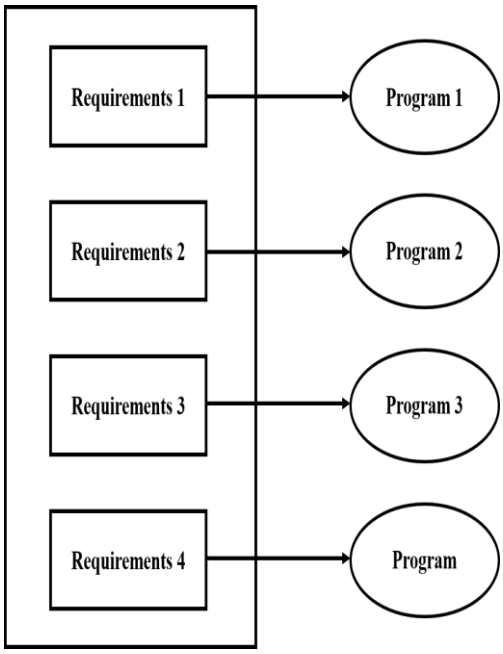


Figure 2.4: Represented that the Loop Testing.

iii. Condition Testing:

In this, it will test all logical conditions for both **true** and **false** values; that is, it will verify for both **if** and **else** condition. For Example:

If (condition)-true

{

}

else-false

{

}

The above program will work fine for both the conditions, which means that if the condition is accurate, and then else should be false and conversely.

Black Box Testing

Black box testing is somewhat of a software-testing that evaluates just at the program's function without looking at its code or structural properties. A customer-stated technical documentation serves as the main foundation for black box testing as shown in below figure. In this procedure, the tester chooses a process, provides an integer number to test its operation, and concludes whether or not the function achieves its intended results. The function meets the requirements if the output is genuine; else, it fails. The test team will evaluate the findings with the software development before moving on to another function to test. Should there be any serious issues after all operations have been tested, the lead programmer is contacted to make the appropriate corrections.

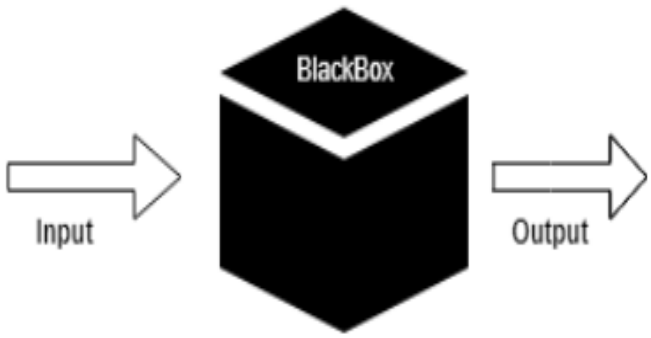


Figure 2.5: Represented that the Block Diagram of Black Box Diagram.

Generic Black Box Testing Procedures

a. Since the black box test is centered on the requirements document, it is looked at first, which shown in Table 2.2.

b. To test if the program is interpreting input values properly or inaccurately, the tester builds positive and undesirable test scenarios by determining valid and invalid model parameters in the second phase.

c. The tester introduces additional test-cases, comprising matrix form, all twosome's tests, equivalent dissection, error estimations, cause-and-effect maps, etc., in the 3rd stage.

d. The deployment of all test cases is component of the 4th phase.

e. The tester compared the predicted output with both the actual output in the 5th phase.

f. If the program seems to have any mistakes, they are motionless as well as established once again in the seventh and ultimate phase [2]

Table 2.2: Methods for Black Box Testing

Sr. No.	Techniques	Procedure
1.	Decision Table Method	The Decision Table Methodology is a systematic way for aggregating different input configurations and the performance of the system that results from them. It is applied to functions that demonstrate a logical relationship to two or more inputs.
2.	Boundary Value Technique	Boundary values, defined as values that include the upper and lower values of a variable, are tested and use the boundary value technique. While inputting a boundary value, it evaluates to see whether the computer is generating the anticipated outcomes.
3.	State Transition Method	The comportment of the software-program while numerous input beliefs are supplied to the same functionality is caught consuming the conditional probability procedure. This is appropriate to the kinds of programmer which afford users a certain handful of legitimate users.
4.	All-pair Testing Technique	Testing of all pairs Technique is implemented to test all imaginable discrete value combinations. This combinational methodology is used to learning outcomes that include input including checkboxes, radio buttons, combo box, text boxes, etc.
5.	Cause-Effect Technique	Cause-Effect Technique emphasizes the connection between a certain objective and all the variables impacting the result. It is based on many false assumptions.
6.	Error Guessing Technique	When using error guessing, the error can really be located using such a specified approach. It is supported by the

		empirical analyst's expertise, and thus the tester utilizes that understanding to infer the software's challenging sections.

Test Procedure

Black-box-testing is a kind of difficult method where the tester-creates unit testing to authorize the performance of the program while retaining comprehensive acquaintance of how the invention performs. There is no need for software developers and e ach test case is constructed by considering the input and the output of a certain function. A tester is simply conscious of the real result of an input; they are uninformed of the method used to get the result. The All-Pair iterative approach, the cause-effect graph procedure, the function specifies protocol, the binary decision technical skill, the boundary performance assessment technique, the state reconfiguration, the error guessing method, the use instance at least method, the nonfunctional and functional method, and the error guessing process are just an a handful of the testing techniques being used black box testing. The course includes thorough discussions of each of these strategies [3].

i. Test Case

The conception of test cases taking the necessities' statement into account. The occupied representations of the program, which include the specifications, strategy criteria, and other characteristics, are often secondhand to produce these test-cases. In directive to get the right response, the test designer indicates both a constructive test-scenario using valid model parameters and a damaging test-premise using broken input tenets. However they may be used for non-functional assessment as well, test belongings are generally generated for purposeful tough [4].

ii. Advantages of Black Box Testing:

a. The tester does not necessitate any extra specialized skills or coding skills in order to perform Black Box Testing.

b. It works well for incorporating the tests in the larger structure.

c. Tests are run out from viewpoint of the user or customer.

d. Duplicating validation checks is uncomplicated.

e. It is used to find discrepancies and ambiguities in technical specifications.

iii. Drawbacks of Black Box Testing:

a. The same tests may be replicated when you put the testing strategy into operation.

b. It is challenging to implement test cases deprived of precise purposeful requirements.

c. Why byzantine inputs at countless difficult segments make it challenging to carry out the test cases.

d. On sometimes, the cause of a test failure cannot be found.

e. Not all of the tender's programs have been confirmed.

f. The mistakes in the rheostat framework are not disclosed.

g. Dealing with a huge illustration intergalactic of inputs may be protracted and time-consuming.

Functional Testing

This type of software testing, known as functional testing, verifies software systems with functional specifications and specifications. Each function of a software system is tested using functional tests, which may involve supplying the correct inputs and comparing the outputs to the functional requirements. Functional testing typically involves "black box" testing and is preoccupied with the application's source code. The user interface, API, database, security, client/server interface and other features of the test have been considered in this program.

i. Testing can be done Manually or Mechanically:

Systems are evaluated against specifications of functional requirements in functional testing, which a subtype of software is testing. Performance testing verifies that the application correctly matches the requirements or specification. This type of test is specifically interested in the success of the processing. It emphasizes the simulation of actual system operation and does not make any assumptions regarding the structure of the system.

In short, it is a type of verification that confirms that each software platform function works as per the requirement and specification. The source code of the platform is not related to this test. Each functionality of something, such as a software system, is tested by providing the correct test inputs, estimating the outputs, and comparing the actual outputs with the desired outputs. This test focuses on verifying the performance of the user interface, running software, and client or server services of the application under test.

ii. Type of functional Testing

The main objective of functional testing is to test the functionality of the component. Functional testing is divided into multiple parts. Here are the following types of functional testing in Figure 2.6 below:

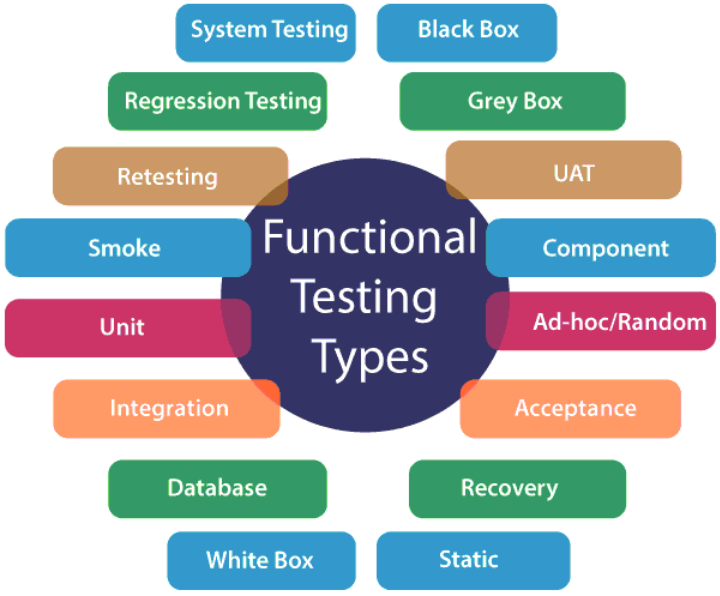


Figure 2.6: Illustrated that the Functional Testing Types.

Unit Testing:

A form of software testing named unit testing checks each individual unit or component of the application. Because unit testing needs to make sure each module is working normally, it also includes functional assessment of the various sets of activities. Unit testing is done by the developer. The application's construction step comprises unit testing.

Smoke Testing:

Smoke testing for functioning evaluation. Only the system's primary functioning is assessed during smoke tests. "Build Verification Testing" is the name given with smoke testing. Smoke testing aims to validate the functionality of the most critical component. For instance, smoke testing, typically confirms that the group discussions correctly, will examine the functionality of the graphical user interface (GUI).

Sanity Testing:

Sanity testing is done to ensure that every aspect of a high-level market scenario is functioning properly. To verify the performance and issues resolved, sanity testing is performed. A bit higher advanced than smoke testing is logical testing. For instance, the login process works properly, all the buttons function as anticipated, and the navigation of the webpage is complete or not after a button is pushed.

Regression Testing:

This kind of testing concentrates on making sure that the present performance of the system won't be negatively affected by the configuration changes. Regression testing focuses around whether all components are working effectively when a defect reappears inside this system after it has been fixed. Test automation focuses on whether the business has been affected.

Integration Testing:

Integration testing linked distinct components and tested those together. This testing's essential aim any issues with how the integrated demonstrated a positive relationship.

iii. Process of Functional Testing

The prime objective of Functional testing is checking the functionalities of the software system. It mainly concentrates as display in Figure 2.7:

a. Mainline Functions: Testing the main functions of an application

b. Basic Usability: It involves basic usability testing of the system. It checks whether a user can freely navigate through the screens without any difficulties.

c. Accessibility: Checks the accessibility of the system for the user

d. Error Conditions: Usage of testing techniques to check for error conditions. It checks whether suitable error messages are displayed.

iv. Advantages of Functional Testing

a. It ensures to deliver a bug-free product.

b. It ensures to deliver a high-quality product.

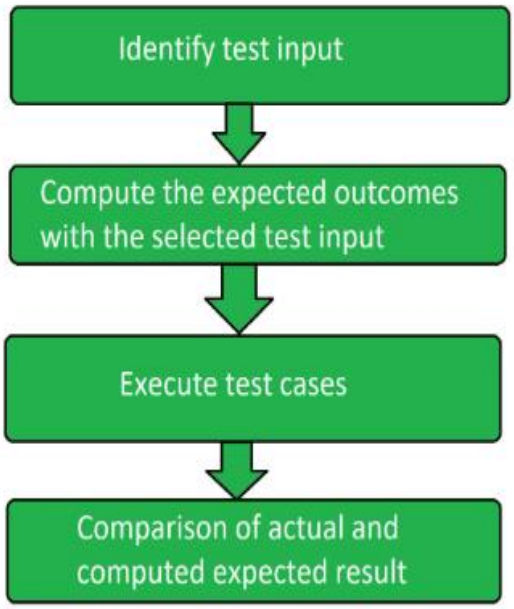


Figure 2.7: Represented that the Process of Functional Testing.

c. No assumptions about the structure of the system.

d. This testing is focused on the specifications as per the customer usage.

v. Disadvantages of Functional Testing

a. There are high chances of performing redundant testing.

b. Logical errors can be missed out in the product.

c. If the requirement is not complete then performing this testing becomes difficult.

Usability Testing

Today, there are several applications in the app store that may assist individuals in their job and where they would leave bad reviews or low scores, which direct a specific product in their favor before a small number of end customers demand or install it. In a nutshell, we should say that one negative review might ruin all of the hard work, talent, as well as strategy that went into creating the company.

This is why usability testing, which plays a significant role and is carried out by test engineers all through software development cycle, enters the equation to address these type of issues. A reported to have the highest of software testing that falls under the umbrella of non-functional testing is package includes.

It is largely used in subscriber interaction design to examine a software product's usability as well as ease of use. Since validation is a thorough process, its execution necessitates for knowledge about the technology.

Usability testing is often performed out from the perspectives of the end user to evaluate whether or not the network performs well.

Figure 2.8: Represented that the Usability Testing.

According to the above Figure 2.8, usability testing assesses how user-friendly, functional, and accurate the system is. This same main goal of doing functional testing is to ensure that the application will be simple for the inevitable end to use while maintaining the functional and commercial criteria provided by the client. The employment of usability testing insures that the created software is simple to navigate without any issues and makes the lives of end users simpler. In other words, usability testing is one of the distinguishing testing methods that reveals imperfections in the way a software product interfaces with its end users. Because of this, it is sometimes referred to as "User Experience (UX) Testing." It assists us in resolving various usability issues associated with a particular website or software, as well as ensuring its efficiency and functioning.

Usability testing is used to certify all of a product's essential aspects, including the ease in which a website can be navigated, as well as its substance and flow, in order to suggest the optimal user experience. Usability testing is most often carried out by real customers rather than the development team given that we already know that the engineering team is the ones that constructed the product. As a result, they miss all those little flaws or faults that influence the customer experience.

The following qualities may be used in usability testing to characterize user-friendliness:

a. Easy to understand

b. Easy to access

c. Look and feel

d. Faster to Access

e. Effective Navigation

f. Good Error Handling

Performance of Usability Testing

We need usability testing that doing so will help in creating systems that do provide excellent user engagements. Usability is not only used only for creating software or internet; it is also used to produce stuff, and customers must feel at ease using your software under the following requirements.

a. The flow of an Application should be good

b. Navigation steps should be clear

c. Content should be simple

d. The layout should be clear

e. Response time

And we can also test the different features in usability testing given as follows:

a. How easy it is using the application

b. How easy to learn application

Features of Usability Testing

The implementation of usability testing helps us to increase the end-user experience of the particular application and software. With the help of usability testing, the software development team can quickly detect several usability errors in the system and fix them quickly. Some other vital features of usability testing are as follows:

a. It is an essential type of non-functional testing technique, which comes under the black-box testing technique in software testing.

b. Usability testing is performed throughout the system and acceptance testing levels.

c. Generally, usability testing is implemented in the early stage of the Software Development Life Cycle (SDLC).

d. The execution of usability testing offers more visibility on the prospects of the end-users.

e. The usability testing makes sure that the software product meets its planned purpose.

f. It also helps us to find many usability errors in the specified software product.

g. Usability testing mainly tests the user-friendliness, usefulness, traceability, usability, and desirability of the final product.

h. It offers direct input on how real-users use the software/application.

The usability testing includes the systematic executions of the product's usability under a measured environment.

Compatibility Testing

This is Non-functional testing which is includes it. Compatibility testing includes reviewing an application's usability across application algorithms, hardware, network, and web browsing platforms.

i. Use Compatibility Testing

Once the product is functional, we take it into development. Since it may be used or accessed by several individuals who use different platforms, we do one round of incompatibility testing to ensure that no problem comes.

ii. Types of Compatibility Tests

Let's look into compatibility testing types

a. **Hardware:** It checks software to be compatible with different hardware configurations.
b. **Operating Systems:** It determines if your technology is compatible with many different operating systems, like Window panes, UNIX, Mac OS, and others.
c. **Software:** It verifies that the technologies you generated is comparable with other software. For addition, the MS Word computer should work with other systems like MS Outlook, MS Excel, and VBA, among others.
d. **Network:** A system's functionality inside a network is analyzed using a variety of characteristics, such as connectivity, operating speed, and volume. Along with the above listed factors, it also assesses applications running on different networks.
e. **Browser:** It examines your webpage compatibility with so many browsers, including Firefox, Google, and Internet Explorer, and some others.
f. **Devices:** It determines if your computer is compatible with a variety devices, including Bluetooth, USB port devices, printers, and scanners.
g. **Mobile:** Verify that your technology is compatible with various mobile operating systems like iOS and android.
h. **Versions of the software:** his involves ensuring that their computer program is compatible with various updates and patches. Test Microsoft Word's reliability with Windows 7, Windows 7 SP1, Windows 7 SP2, and Windows 7 SP3, for example.

iii. Backward Compatibility Testing

Hardware compatibility testing is a way to check how domestically made hardware or software behaves and works with earlier incarnations of the same gear or software. Since all the adjustments from the prior versions are known, backward integration testing is considerably simpler predictable.

iv. Forward Compatibility Testing

Forward Interoperability Testing is a procedure to evaluate how newly produced hardware or software responds and works with older models of the same gear or software. It might be problematic to anticipate forward testing procedures since it is unknown what adjustments will be implemented in older editions.

Tools for Compatibility Testing

a. Browser Stack:

Browser Compatibility Testing: This tool helps a software tester to crisscross application in changed browsers.

a. Virtual Desktops:

Operating System Compatibility: This enables the functioning of programs as virtual machines throughout several operating systems. It is it possible to connect "n" systems and examine the results.

Process of Compatibility Testing

b. Identifying the range of operating systems or platforms that the application is anticipated to run upon is the first step in the comparability testing process.

c. The tester should be knowledgeable enough with the frameworks, software, and hardware to understanding the intended behaviours of the system in diverse positions.

d. The environment must also be prepared for testing with multiple networks, devices, and networks to verify that your application performs admirably in various setups.

Document relevant bugs. Correct the flaws. Keep testing once defect-fixing is validated [4].

Non-Functional Testing

Software testing of the non-functional kind is carried out to verify that the software meets its non-functional criteria. It checks to ensure that the system is acting in accordance with the guidelines or not. It examines every component that is not really examined during testing process.

Software analysis techniques called "non-functional testing" examine the system's non-functional qualities.

Non-functional testing is a sort of program development used to examine a programming application's non-functional features. It is intended to assess a system's readiness thus according nonfunctional criteria that test plan never takes into account. Both functional and non- functional testing are crucial.

ii. Objectives of Non-functional Testing

Non-functional testing's goal is to:

a. To improve the manufacturer's use, effectiveness, unwavering quality, and portability.

b. To assist in lowering industrial risk associated with the manufacturer's non-functional component.

c. To assist in lowering the cost of something like the product's non-functional components.

d. To improve the product's methodologies for installation, use, and supervision.

e. To gather data and generate indicators for use in internal research and development efforts.

f. To advance and broaden knowledge about current performance and help and technology.

Non-Functional Testing Techniques

a. Compatibility Testing:

A sort of testing to make that somehow a system or piece of software is compatible with several other systems or software application.

b. Compliance Testing:

A special kind of testing to confirm that a software programme or infrastructure complies with a specified standards and requirements, like HIPAA or Sarbanes-Oxley.

c. Endurance Testing:

A sort of testing to determine that a system or software implementation can withstand a prolonged, long-term load.

d. Load Testing:

A method of testing to try and ensure a system or piece of software can manage a large number of individuals or transactions.

e. Recovery Testing:

A sort of testing to confirm that a system or software programs can be repaired after a problem or data loss.

f. Security Testing:

A sort of testing to evaluate that a system or software product is protected against incursion or assault.

g. Scalability Testing:

A sort of testing undertaken to ensure that a system or piece of software can be scaled either up or down to accommodate changing demands.

h. Stress Testing:

A sort of testing to assure a system or piece of software can tolerate a very high load.

i. Volume Testing:

A method of testing to try and ensure a system or software product can accommodate a lot of data.

Difference between Functional Testing and Non-functional Testing are elaborated through the Table 2.3. Which is holds the sequential order.

Table 2.3: Difference between Functional Testing and Non-functional Testing

Sr. No.	Functional Testing	Non-functional Testing
1.	Utilizing the functional requirements that the customer has furnished, functional testing checks the platform against the requirements specification.	Non-functional testing examines the software system's scalability, performance, and other non-functional elements.

2.	First, functional testing is carried out.	Functional testing ought to come before non-functional testing.
3.	Functional testing may be carried either manually or automatically using various techniques.	After functional testing, non-functional testing should be carried out. Utilizing tools will help with this testing.
4.	Functional testing is a process that starts with business requirements.	Non-functional testing uses performance factors like speed and scalability as inputs.
5.	Functional testing outlines the capabilities of the product.	Nonfunctional testing describes the product's usability.
6.	Simple manual testing	Manual Testing is challenging

iii. Performance Testing

Software testing in the form of performance testing verifies that software programmes function as envisioned under the market demands. It is a method for testing technologies to ascertain how hypersensitive, reactive, and stable they are during a particular workload.

The practise of reviewing a product's capabilities as well as quality is called performance benchmarking.

It is a testing technique for assessing the system's speed, stability, and stability under ability to apply knowledge. Performance testing seems to be another name for performance testing.

Performance Testing Attributes:

- **Speed:** It establishes how quickly the software product reacts.
- **Scalability:** It establishes how much load the software product can support at once.
- **Stability:** It establishes if the program is stable under different workloads.
- **Reliability:** It establishes whether or not the program is secure.

Objective of Performance Testing:

- Performance testing's goal is to get rid of performance bottlenecks.
- It identifies areas that need to be addressed before the product is introduced to the market.
- The goal of performance testing is to speed up software.
- The goal of performance testing is to create stable and dependable software.

Types of Performance Testing:

i. ***Load testing*****:**

It evaluates the performance of the product under realistic user loads. Prior to the release of the software product in the market, the goal is to detect performance problems.

ii. *Stress testing*:

It entails putting a product through extensive evaluation to assess how it responds to congested areas. Finding the software manufacturer's breaking point is the goal.

iii. *Endurance testing*:

It is done to make sure the programs can sustain the anticipated pressure over a longer period of time.

iv. *Spike testing*:

It evaluates how the product adjusts to unexpectedly high user consumption spikes.

v. *Volume testing*:

In volume testing, a lot of data typically recorded in a spreadsheet and the general behavior of something like the software system is examined. The goal is to evaluate the functionality of the product with actual database volumes.

vi. *Scalability testing*:

Software applications' abilities to scale up in response to an increase in user demand is tested for portability. It aids in the budgeting of software system capacity additions.

Advantages of Performance Testing:

- Performance testing verifies the system's responsiveness, load capacity, correctness, and other attributes.
- If anything goes wrong, it locates, keeps an eye on, and fixes the problems.
- It guarantees the software's excellent optimization and permits a big number of concurrent users.
- It guarantees the happiness of both clients and final consumers.

Disadvantages of Performance Testing:

- Users sometimes may encounter performance problems in a live environment.
- Team members should have a high degree of understanding while building test scripts or test cases in the automation tool.
- The test cases or test scripts should be very proficiently debugged by team members.
- Low performance in the actual world may result in a significant drop in user numbers.

iv. Scalability Testing

Scalability testing is a sort of non-functional testing when the performance of a software application, system, infrastructure, or process is appraised in terms of its capacity to scalability up or scale down the quantities of user requests, as well as various other performance-related characteristics. It may be done at the database, programming, or hardware levels. This same capacity of a network, system, application, product, or process to fulfil the function when changes are introduced in the size or volume of that system to suit a rising requirement is related

to as scalability. It guarantee that a software product can sustain the planned rise in user traffic, bandwidth usage, transaction counts periodicity, and many other factors. It analyses how well the system, procedures, or information can support an expanding requirement.

Scalability testing evaluates the point at which a software or hardware product stops scaling and identifies the cause. Depending on the application, several criteria are utilized for all of this assessment. For instance, the number of users, Computational power, and network usage all go into the scalability evaluation of a web page, while the volume of responses handled determines the adaptability of a web server.

Objective of Scalability Testing:

The objective of scalability testing is:

- To learn how applications extend as the workload grows.
- To determine its user limit for the software item.
- To evaluate client-side depreciation with load and customer journey.
- To assess the server-overall side's degeneration in terms of reliability.

Scalability Testing Attributes:

i. Response Time:

The amount of time that happens between a user's request and an application's acknowledgment is known as the response time. Depending on the application's user engagement, response time may go up or down. In general, an application's response time increases slower the more users it has to serve. Applications with either a faster reaction time typically regarded as having outstanding quality.

ii. Throughput:

The quantity of requests delivered by the application during a specific amount of time is referred to as velocity. Different applications measure it differently; for example, some web application evaluates the number of user requests handled in a short period of time, while a software program counts the number of queries performed in an amount of time.

iii. Performance measurement with number of users:

Depending on the kind of service, it is always tested to figure out how many people it can handle before malfunctioning or going into a busy standby state.

iv. Threshold load:

The amount of requests or operations the application can handle while maintaining the acceptable throughput is known as the borderline load.

v. CPU Usage:

CPU Usage is an assessment of the CPU's activity during the performance of application code. Basically, it is stated in terms of the megahertz unit.

vi. Memory Usage:

Memory use is a representation of how much memory a program uses to carry out a job. In essence, it also is quantified in terms of bytes.

vii. Network Usage:

The bandwidth consumed used by a test application is defined as network consumption. Bytes generated per second, frames per instant, segments received and transmitted per instant, etc. are used as measurements units.

Types of Scalability Testing:

Scalability testing is classified into two parts, which are as follows display in Figure 2.9:

- Upward scalability testing
- Downward scalability testing

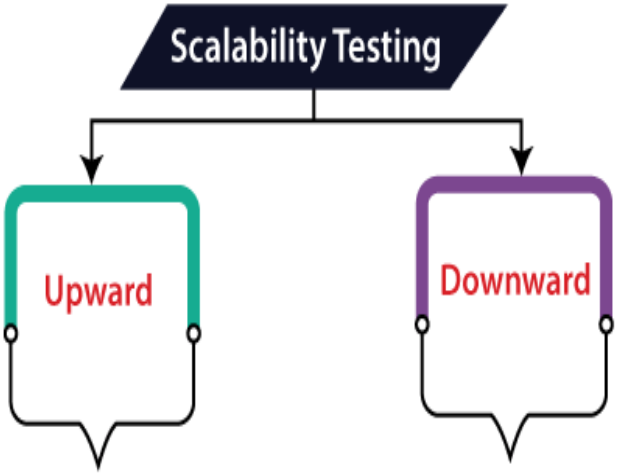


Figure 2.9: Represented that the Scalability Testing.

➢ **Upward scalability testing**

The upward scalability testing is used to expand the number of users on a specific scale until we got a crash point. It is mainly used to identify the maximum capacity of an application.

➢ **Downward scalability testing**

Another type of scalability testing is downward scalability testing. When the load testing is not passed, we will use the downward scalability testing and then start decreasing the number of users in a particular interval until the goal is achieved.

Therefore, we can quickly identify the bottleneck (bug) by performing downward scalability testing.

- **The Precondition for Scalability Testing**

For the scalability testing, the test strategy can be changed based on the type of application being tested. For instance, if a database is related to an application, then the testing constraints will be the database and the number of users. We have some default precondition available for scalability testing, which is as follows as Figure 2.10:

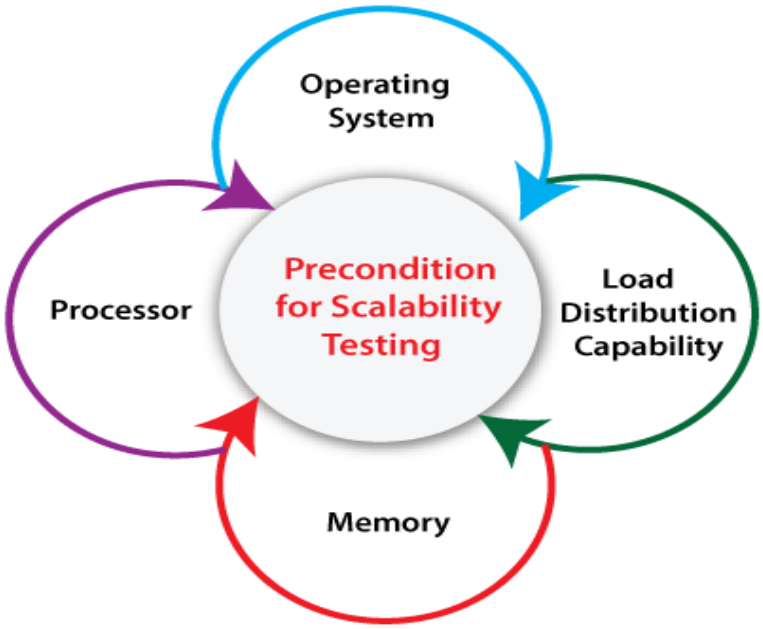


Figure 2.10: Represented that the Precondition for Scalability Testing.

- **Operating System**

If we want to perform the scalability testing, we need to verify what operating systems are prepared by the load generation managers and load test master.

- **Load Distribution Capability**

It is used to analysis if the load testing tool allows the load to be created from several devices and measured from an essential point.

- **Memory**

Before performing the scalability testing, we must analyze how much memory would be sufficient for the virtual user and load test master.

- **Processor**

We need to analyze what type of CPU is required for the load test master and virtual users before executing the scalability testing.

Feature of Scalability testing some of the vital components of scalability testing are listed below; let see them one by one:

a. Throughput

b. Performance measurement with many users

c. Memory Usage

d. CPU Usage

e. Network Usage

f. Response Time

Throughput

a. The throughput feature is used to specify the amount of work implemented by the application in the given time.

b. The throughput can change from one application to another.

c. For example, in a database application, it is sustained in several commands managed in a unit of time. In contrast, it is uniform in the number of user requests handled in a unit time in a web application.

Performance

a. The next feature of scalability testing is performance, which is used to check the user's repetitively collective load and request under the webserver and repose of the system.

b. In other words, we can say that the performance of the application depends on its type as it is tested continuously for several users, which can support without its failure or backup condition.

Memory Usage

a. In scalability testing, Memory usage is one of the resource utilizations used to sustain the memory expended for performing a task by an application.

b. Typically, the memory usage is calculated in respect of the unit bytes.

CPU Usage

a. Another resource utilization under scalability testing is CPU usages, which is used to calculated the CPU utilization while performing application code commands.

b. Normally, the CPU usage is calculated in respect of the unit Megahertz.

Network Usage

a. It is used to carry the bandwidth consumed by an application under test.

b. The network usage is calculated in terms of frames received per second, and bytes received per second, segments received and sent per second, etc.

Response Time

a. It is the time used up between the application's response and the user's request.

b. In other words, we can say that the response time checks how fast the system or the application response to user or other application requests.

c. It may enhance or reduce the time based on different user loads on the application.

d. Usually, the response time of an application reduces as the user load is enhanced.

Steps involved in Scalability Testing:

Following are the steps involve in the scalability testing:

a. Establish a repeatable procedure for carrying out the scalability test.

b. Establish the standards for the scalability test.

c. Establish the software tools necessary to run the test.

d. Set up the hardware needed to carry out the scalability test and the testing environment.

e. Write a visual script and check it.

f. Construct the load test scenarios and validate them.

g. Carry out the exam.

h. Assess the outcome.

i. Produce the necessary report.

Advantages of Scalability Testing:

a. It makes the product more accessible.

b. It recognizes performance problems, including problems with web page loading.

c. It locates and resolves faults with the product sooner, which saves a lot of time.

d. It guarantees the end-user experience while handling the particular load. Customer satisfaction is provided.

e. It facilitates efficient tool use tracking.

Disadvantages of Scalability Testing:

a. Some automated techniques used for Scalability testing are more expensive, which eventually raises the budget of the product.

b. It sometimes fails to detect the functional flaws or issues in the product.

c. Team members that participate in this testing approach should be highly skilled testers.

d. Testing some product components may take longer than anticipated;

e. Unexpected findings may also emerge once the product is introduced to the consumer environment.

CHAPTER 3

AN INTRODUCTION OF SYSTEM SECURITY IN SOFTWARE TESTING

Jayaprakash B, Assistant Professor
Department of Computer Science & IT, School of Sciences, Jain (Deemed-to-be University), Bangalore-27, India
Email Id- b.jayaprakash@jainuniversity.ac.in

Testing a completely assimilated software system is part of testing the system. Software is often integrated into personal computers; nevertheless, software is essentially one component of a computer system. To produce a comprehensive computer-system, the software is built in components and then interfaced with electronics and other applications [1]. To put this another way, a computer system comprises a set of software that can carry out a variety of functions, but only software can do so subsequently it has to associate with appropriate hardware. Testing process is a assortment of several types of tests aimed at putting an integrated software system through its paces and validate it against standards.

System testing is the technique of examining an application's as well as piece of software's important in terms. We test the product as a complete solution and travel (go through) together all required modules of an applications to see whether the final features or just the final business function as designed. The testing environment is end-to-end examination where it is comparable to that same manufacturing process [2].

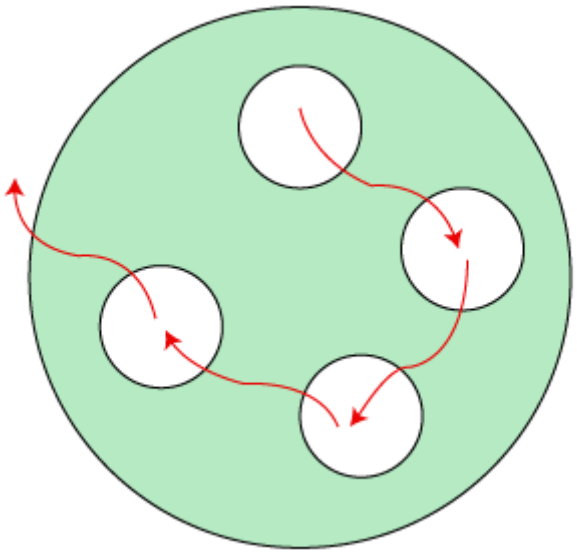


Figure 3.1: Displayed that the System Security.

Unit-testing, integration-testing, system testing, and formal verification are the four layers of software-testing that are utilized.

Testing procedure is used to test an entire system, Assimilation testing is used to test a combination of software components, Testing phase is used to test a single component of software, and Acceptance testing is carried out to determine if prerequisites are acceptable as display in Figure 3.1. The third testing stage that we've been talking about here refers to type of testing [3].

Example of System Testing

If we launch an application, let's say www.rediff.com, we could discover that an advertising is displayed at top of the site and that it remains there for a small period of time before removing. The Advertisement Managerial System creates these forms of advertisements (AMS) [4]. Now, we will complete system-testing for this category of ground. The below submission everything in the succeeding manner:

a. Suppose Amazon wanted to have a promotion ad broadcast on the Rediff front page for India on January 26 at exactly 10:00 AM.

b. After that, the marketing manager accesses the website and files an application for an advertising that used the above-mentioned day's date.

c. He or she applies and includes a file, most likely a video file or indeed a picture file of the AD.

d. The Rediffmail AMS manager logs through into program the following morning to confirm the existing Ad request.

e. The AMS administrator will look at any current Amazon ad requests before determining if there is spare capacity on the requested day though time.

f. If space is available, the cost to broadcast the advertisement is predicted to be $15 per second, or around $150, for a total of 10 seconds.

g. The AMS management clicks on the registration fee and sends the Amazon manager the advance payment together with the predicted sum.

h. After the Amazon had become the payment request and logs through into Ad status, he or she clicks upon this Submit and Pay button to finalize the payment.

i. As soon as the Rediff AMs administrator receives the funds, he or she will schedule the announcement for the particular day and time upon that Rediffmail homepage.

Example of System Testing

a. System testing, which assesses the whole requirements of the system, offers complete assurance of performance of the system.

b. System software infrastructure and business needs are tested.

c. It helps eliminate bugs and live problems however after production.

To assess the differential expression between newly added as well as existing features, system testing combines both an existing system and a new process to input the same data into this one. This allows the user to grasp overall advantages of the newly added the system's functions. [4].

Security Testing

Software examination includes security testing, that is employed to find software application flaws, vulnerabilities, or threats in addition to to assist us thwart malicious outside intrusions and ensure the security of our application programs. Finding all of the database's possible ambiguities and vulnerabilities has been the main goal of security testing, because keeps the product functional. Protection testing enables us to recognize all potential security risks and supports the programmer

in correcting certain flaws. It is a testing method that examines if the data will be secure and keeps the application functioning.

Principle of Security Testing

Here, we will discuss the following aspects of security testing:

a. Availability

b. Integrity

c. Authorization

d. Confidentiality

e. Authentication

f. Non-repudiation

i. Availability

This ensures that the data and statements services will be provided whenever we need them by demanding that the data be kept on account by an authorized individual.

ii. Integrity

In doing so, we will safeguard information that an undocumented immigrant modified. Integrity's main goal is to grant the receiver with control over the information that the system requires. Integrity's main goal is to grant the receiver access to the data that the system requires. The secrecy structures and integrity technologies often use some of the same underlying strategies. Therefore, instead of encrypting every connection, they often incorporate the data transmission data to establish the foundation of an algorithmic check. Furthermore, make sure that appropriate figures is sent from one service to the next.

iii. Authorization

It involves determining whether a person is allowed to both execute an activity and receive assistance as display in Figure 3.2. Access control is a nice demonstration of licensing.

Figure 3.2: Represented that the Process of Authorization.

iv. Confidentiality

It is the only method we can ensure the security of our data, it is a security procedure that delays the leak of data from outsiders.

v. Authentication

In order to provide access to the system or private information, it is important to trace the source of a product and validate a person's uniqueness. The block diagram are display in Figure 3.3.

Figure 3.3: Represented that the Identification Authentication Process.

vi. Non- repudiation

It is a term used to refer to digital security, and it provides assurance that neither the sender nor the receiver of a communication may dispute having transmitted or received the message. The non-repudiation principle is used to confirm that a communication has been sent and received by the individual claiming to have done so.

Cookies

A cookie is a little piece of information that a web server saves to the user's hard drive as a text file. The search engine then makes use of this knowledge to request data from that machine. The cookie often contains individual data or information that is modified and used to interact across websites. Cookies are simply used to authenticate a person, and they serve the purpose of track their path around a website's pages. The communications between a web browser and an internet server is stateless.

Cookies testing

Software testing called "cookie testing" looks at the cookies each web browser creates. Also on user's (client's) hard drive, a text file designated a cookie is where the web server retains a little quantity of information.

This information is sent back to the server each time the browser asks for a page from it. Cookies often provide specific user information or data that has been shared across websites. The image that follows depicts cookies from different websites. In other regards, cookies are just a user's identifier and they are employed to track user movement around for a website's pages. A cookie's goal is to allow rapid communication between users and websites. A shopping cart, a personalized experience, user identification, marketing, user sessions, and many other applications may all be provided through cookies.

Saved the Cookies

The critical check list and detailed instructions for testing cookies on a website are provided below.

Cookies must be disabled:

Try to use the website's primary functions while disabling all cookies.

Cookies tainted:

Manually change the cookie's settings in a notepad with some random ones.

Encrypting Cookies:

Information like passwords and usernames need to be encrypted before being sent to our computer.

Cookie testing with a variety of browsers:

Check that your website is properly writing cookies on a different browser as intended by doing cookie testing across a number of browsers.

Cookies can be Used in the Following Ways:

To Implement the Shopping Cart:

Cookies are used to keep an online ordering system running smoothly. Cookies keep track of what the user wishes to purchase. What if a consumer adds certain goods to their shopping cart and then decides not to buy them this time because of whatever reason and shuts the browser window? When the same user returns to the buy page, he will be able to see all of the goods he put in his shopping basket on his previous visit.

a. Personalized Sites:

When a person views a certain website, they are asked which other pages they do not wish to see. User preferences are saved in a cookie, and those pages are not displayed to the user until he is online.

b. User Tracking:

Counting the number of unique visitors who are online at any one moment.

c. Advertising:

Some businesses use cookies to display advertising on user computers. These adverts are controlled by cookies. When and how should advertisements be shown? What is the user's point of view? What keywords do they look upon the site? Cookies may be used to keep track of all of these things.

d. User Sessions:

Using a user ID and password, cookies may monitor user sessions to a certain domain.

Drawbacks of Cookies

While writing a Cookie is a great way to keep users engaged, if the user has set their browser to warn them before writing any Cookies or has completely disabled cookies, the site containing the Cookie will be completely disabled and unable to perform any operations, resulting in a loss of site traffic.

This may be turned off or on in your browser's settings. For Google Chrome, for example, go to Settings -> Advanced -> Content Settings -> Cookies. You may apply a cookie policy to all websites or set it up for specific ones.

In addition to browser settings, changes in EU and US regulations require developers to notify users that cookies are being used on their websites. Compliance with such new rules should be included in test scenarios for specific areas.

a. Too many Cookies:

If you are writing too many cookies on every page navigation and the user has enabled the option to warn before writing the Cookie, this may turn away users from your site.

b. Security Concerns:

Personal information about users is sometimes saved in Cookies, and if the Cookie is hacked, the hacker can access your personal information. Even damaged cookies can be read by several websites, posing security risks.

c. Sensitive Information:

Some websites may write and keep sensitive information about you in cookies, which is not permitted owing to privacy concerns. This should be sufficient to understand what Cookies are.

d. Recovery Test

Recovery Testing is software testing technique which verifies software's ability to recover from failures like software/hardware crashes, network failures etc. The purpose of Recovery Testing is to determine whether software operations can be continued after disaster or integrity loss. Recovery testing involves reverting back software to the point where integrity was known and reprocessing transactions to the failure point.

Recovery Testing Example

While a network request is being processed by an application, disconnect the connecting cable from the device.

a. Reconnect the cable after a time and check to see whether the program can still receive data from the area where the network connection was lost.
b. Restart the computer when a browser has a particular number of open tabs to check whether it can restore all of them.

In software engineering, recoverability testing is a kind of non-functional testing. Non-functional testing looks at aspects of the software that may not be related to a specific function or user action, including scalability or security.

The number of days required for recovery varies depending on:

- The number of restart points
- The quantity of submissions
- The tools for rehabilitation that are available, as well as the skills and knowledge of people carrying out recovery activities.

Recovery testing should be carried out methodically when there are several failures, which entails finishing recovery testing for one segment before going on to the next. The testing is carried out by qualified testers. Before recovery testing, a significant quantity of backup data is stored in secure locations. This is done to ensure that the business can operate in the case of an emergency.

Testing the Installation:

Installation verification is the practice of testing the appropriate steps to install a usable software system. This installation includes all the steps for testing, total or partial updates, and installing

and wiping additional features. The installation testing examines whether or not the computer software has been installed correctly and that all of its built-in functions. It is also known as implementation validation and is often carried out as the final stage.

a. Testing based on activity

b. Carried out during operational Acceptance testing.

c. Carried out by configuration management and software testing engineers.

d. Delivers the best possible user experience.

e. Aids in the discovery and detection of installation-related issues.

f. The last part of the STLC includes installation testing.

The installation testing procedure sometimes encounters difficulties, and it often degenerates into chaos.

a. Many conditions for validity
b. The item needs to be tested with various configurations.

The requirements varies across platforms just as the software installation procedure is evaluated for a variety of factors, and automated processes are employed to save time consumption.

Performing Testing for installation:

The developers are used to test the installation. The company's development offers both the manual and access to instal packages. The testers essentially get an understanding of the testable and non-testable components from these items.

The development team gets alerted if there are any problems throughout the process. Making the simplest and easiest manual possible is the primary goal of the situation in order to get the greatest results.

Installer testing objectives:

The major goal is to make sure there are no obstacles that limit the software's effectiveness and hinder end users from utilizing it. This retrieves the bug and errors. It guarantees that no issues with the various platforms will arise. It helps in confirming that the following software was distributed correctly to the specific area.

a. Many forms of attendance:

b. Hushed installation

c. Installation that was seen

d. Installation that is unattended

e. Establishing a network

f. Pristine installation

g. Installing automatically

1.4.1. Advantages to installation testing

a. The primary benefit is that it validates software and app designs at the most fundamental level of test performance.

b. It is a very important component of STLC that aids in maintaining the required standard.

c. It's an efficient way to quickly determine the program version.

d. The developer may enhance the program or software with the aid of installation testing's higher output outcomes.

Drawback to installation testing

a. Failures may occur as a result of both external events and defects in the code, making the procedure time-consuming and exhausting.

b. The process of running test cases takes time, particularly when testing is being done during installation.

c. The test case-driven approach completely determines the output or results.

Risk Based Testing

Software testing of the Risk Based Testing (RBT) kind is founded on the likelihood of risk. It capable of being accepted the risk in light of factors such as software complexity, the profitability of the company, use practices, and potential danger areas. Testing of software program aspects and functionalities that are most important and likely to produce flaws is given priority in risk-based testing.

Risk refers to the chance of an unanticipated incident that could have favorably or unfavorably impact on the quantifiable desired outcomes of a project. Developments from the past, those actually occurring now, or potential events in the ahead might all be included. Unexpected unforeseen circumstances may impair a project's financial, marketing, technical, and needed services.

Risks can be positive or negative.

a. Opportunities that are supported by previous risks aid in the development of businesses. For instance, spending money into a new development, altering corporate operations, or creating new goods.

b. Negative risks are thought to as threats, and for a project and being successful, suggestions to eliminate or reduce them must be implemented.

Implementation of Risk Based Testing

Risk based testing can be implemented in

a. Projects with constraint on time, resources, or cash, etc.

b. Projects where vulnerabilities to SQL injection attacks may be recognized using risk-based methodology

c. The critical variable for secure cloud computing.

d. There are significant dangers in new projects, such as minimal awareness of the skills being used or little industry specific experience.

e. Progressive and iterative simulation strategies, etc.

Risk Management Process

i. Risk Identification

Risk workshops, checklists, discussion groups, interviews, the Delphi methodology, control charts, lessons learned from past projects, root cause analyses, and consults with domain and subject matter professionals are all methods for identifying risks. The Risk Register is a spreadsheet that contains a list of acknowledged dangers, possible treatments, and underlying causes. Throughout the course of a project, it is used to monitor and oversee risks, including hazards and opportunities. Both positive and undesirable risks may be managed using proposed mitigation techniques. In risk planning, the risk breakdown structure is important. The Risk Breakdown hierarchy would aid in locating the developer's risk-prone sections and provide streamlined appraisal and risk monitoring. It helps in giving risk management actions enough resources and attention. It also helps in classifying a number of possible sources from which development hazards might develop.

ii. Risk Analysis (Includes Quantitative and Qualitative Analysis)

After compiling a list of prospective hazards, the subsequent stage is to examine them and characterize the risks according to potential importance. Utilizing a risk matrix is an example of qualitative risk analysis methodologies (covered in the next section). This method is employed to determine the risk's effect and likelihood.

iii. Risk Response Planning

Using the study, we can determine if the concerns call for a reaction. For instance, certain concerns will need to be addressed inside this project design, while others will need to be treated in the performance measurement, and yet others won't be necessary to be addressed at all. The risk owner is in charge of coming up with strategies to lessen the likelihood and effect of the various motivations. Risk response measures like risk mitigation are used to reduce the undesirable impact of potential hazards. This may be performed by getting rid of the threats or lessening them to a respectable level.

iv. Risk Contingency

Contingency may be described as the possible development of a hazard, but the outcome is unpredictable or unexpected. The implementation plan or backup preparations for the worst-case eventualities are other names for a contingency plan. In plenty of other words, it decides what may be done in the case that an unforeseeable incident occurs.

v. Risk Monitoring and Control

To track the risks that have been detected, to keep an eye on possible risks, to find new risks, to revise the risk register, to investigate the reasons of change, to carry out risk management plans,

to keep an eye on risk triggers, etc., risk management monitoring processes typically utilized. Analyze how well they perform in lowering vulnerabilities.

vi. Risk Based Testing Approach to the System Test

Risk Based Testing Approach to the System Test are mention in below Figure 3.4:

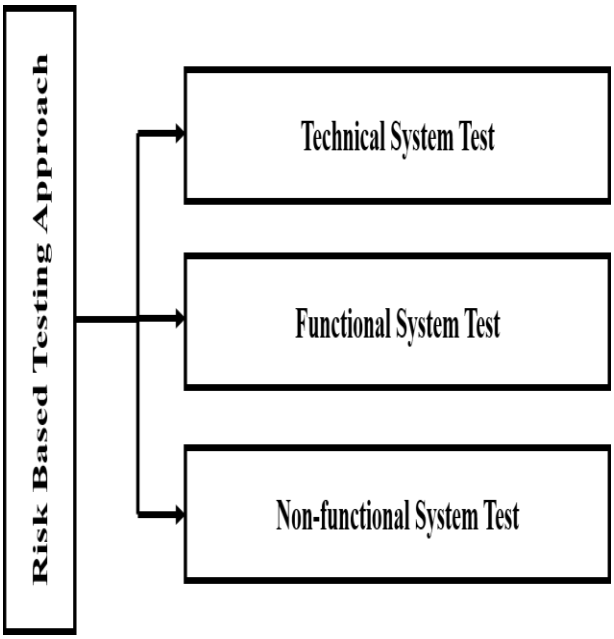


Figure 3.4: Represented that the Risk Based Testing Approach.

- **Technical System Test**:

This is referred to as environment test and integration test. Environment test includes testing in development, testing, and the production environment.

- **Functional System Test**:

Testing of all functionalities, features, programs, modules. The purpose of this test is to evaluate if the system meets its specified requirements.

- **Non-functional System Test**:

Environment test and integration test are words used to describe this. Testing in the developmental, testing, and performance testing are all included in environment testing. Testing of all capabilities, features, modules, and technology. This test's goal is to determine if the program meets the criteria which have been given to it. Performing load tests, stress tests, reconfiguration tests, security tests, backup and restore processes, and documenting testing for non-functional needs system, operation and installation documentation.

vii. Benefits of Risk Based Testing

The benefits of Risk Based Testing is given below

a. Enhanced productivity and cost savings

b. Better addressable market, faster time to market, and timely delivery.

c. Increased service quality

d. Better quality since all of the software's crucial features have really been put to the test.

e. Clearly explains test execution information. We are aware of what has been tested and what has not even using this method.

f. The most economical and efficient strategy to reduce the residual risk after release is to allocate test effort based on risk assessment.

g. The company can determine the residual degree of quality risk during test execution and make informed release choices thanks to test result assessment based on risk analysis.

h. Enhanced testing using well defined risk assessment techniques.

i. Greater client satisfaction because customers were involved, there was effective reporting, and progress was tracked.

j. Early identification of possible trouble spots. To solve these issues, practical preventative actions might be used.

k. Ongoing risk monitoring and assessments throughout the whole of the project's lifetime assists in recognizing and resolving risks and addressing problems that might jeopardies the completion of the project's overall goals and objectives.

CHAPTER 4

INTRODUCTION OF AUTOMATION TESTING

Dr. Santosh S Chowhan, Assistant Professor
Department of Data Science & Analytics, School of Sciences, Jain (Deemed-to-be University), Bangalore-27, India
Email Id- santosh.sc@jainuniversity.ac.in

Automation testing is a different strategy to testing software development that employs specialized tools to run test scripts unattended as display in Figure 4.1. It is the most accepted technique to maximize software testing's performance, production, and test coverage. We may conveniently approach the experimental results, manage the test implementation, and measure the actual output against the accuracy and fairness with the assistance of an automated software tool.

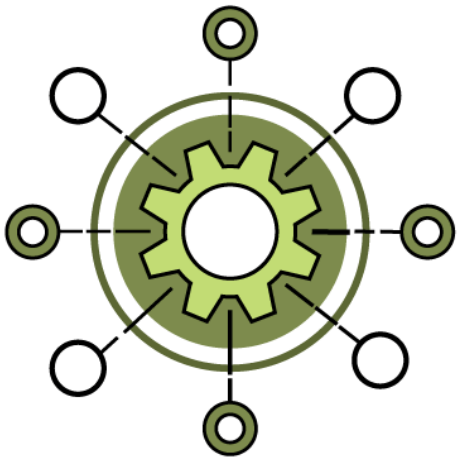


Figure 4.1: Represented that the Automation Testing.

The test automation engineer will construct the test script or utilize the open source automation tools to execute the application during automation testing. In contrast, the functionality testing will create the unit testing and build the program based on them when conducting manual testing. The functionality testing may carry out repetitive duties and other related jobs using test automation. Repeating the repetitious take frequently is a tiresome procedure in testing process. In other words, the primary goal of system testing is to replace manual productive labor with automated methods or tools. The exploratory testing phase of the automatic test process takes up less time than upgrading the test scripts, which improves the overall application performance.

Need to Perform Automation Testing

a. Although automated testing equips us with a smooth implementation with less work and time, it is required in software testing to test the application's functionality.

b. Because they are not totally aware of the automated testing methodology, several firms continue to examine applications exclusively through manual methods.

c. So they're now using test automation in their software development process, they are aware of automation tools.

d. We needed to spend a fair amount of money and effort to execute the automation tools.

The following benefits of automating testing are just a few of the many that we receive [3]:

a. Reusability

b. Consistency

c. Running tests anytime (24/7)

d. Early Bug detection

e. Less Human Resources

i. Reusability:

In automation tools, we may reuse the automated tests rather than always writing new ones. Moreover, we may repeat the prior processes' exact details.

ii. Consistency:

Automation testing is smoother and more reliable than testing on humans when it comes to doing the normal, tedious tasks which cannot be missed yet may result in errors while checked physically.

iii. Running Tests 24/7:

With automated testing, we can able to begin the system testing whenever and from anywhere and even if we do not have many options or the chance to buy them, we can still complete that task remotely.

iv. Early Bug Detection:

By using automation tools, we can quickly find the essential flaws within the first phases of the software design process. Additionally, it enables us to resolve these problems for less dollars and in less working time.

v. Less Human Resources:

Instead than having numerous employees run the tiresome test cases again, we need an automated testing engineer who can create the test scripts necessary to automate our tests.

Automation Testing Methodologies

The three diverse approaches and approaches that help compensate automation testing will aid the design involving in raising the excellence of the software product [4].

- GUI Testing
- Code-Driven
- Test Automation Framework

Now, let's understand the different approaches of automation testing one by one:

i. GUI (Graphical user interface) Testing:

This method allows us to construct the computer program that has GUIs. so that the professional developers on the automation tests may acquire and continuously assess online behavior. We are aware that test cases may very well be developed in a range of programming languages, including Java, C#, Python, and Perl.

ii. Code-Driven:

The adaptive capacity employed in automated testing is considered code-driven technique. In this methodology, the test engineer will mostly emphasis on test case execution to establish if various code components are performing in accordance with the specified specification or not. As a result, it is a widely popular strategy in development of agile software.

iii. Test Automation Framework:

Test automation infrastructure is another strategy in automation. A set of guidelines called the automation testing framework is utilized to get useful results from automated software activities. Similar to that, it puts altogether reusable modules, function modules, object details, and training dataset sources.

Automation Testing Process

The automated testing method is an organized way to plan and execute out testing tasks such that absolute maximum coverage is achieved with both the shortest amount of resources. The test's framework includes a sequential procedure that supports the specific, sophisticated, and interconnected operations needed to complete the objective. The automation testing process completed in Figure 4.2 and explain in the following steps [5].

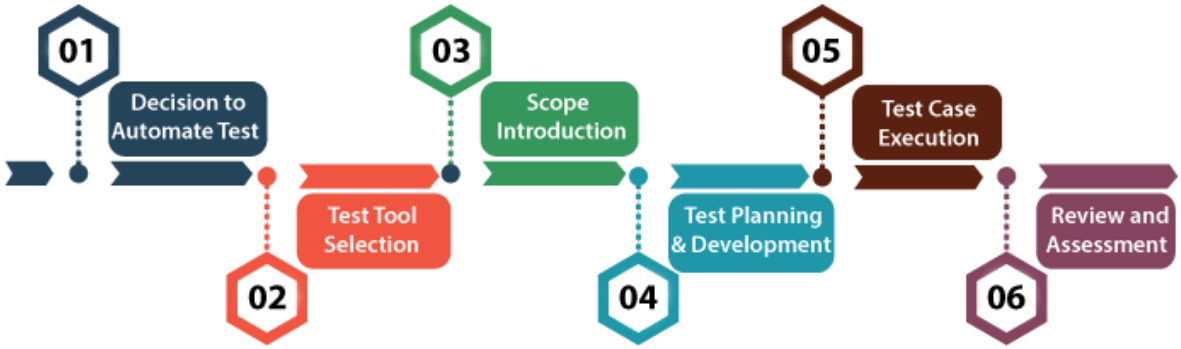


Figure 4.1: Display that the Process of Automation Testing.

i. Decision to Automation Testing:

It is the initial stage of the technique for automated tools, and at this stage, the key purposes of the testing team are to control test expectancies and identify any possible advantage of using automated testing adequately. Organizations must deal with a variety of hurdles while implementing testing activities, some of which are described below:

- Testing tool professionals are needed for automated tools, thus hiring a testing equipment expertise is the first consideration.
- Selecting the ideal technology for evaluating a certain operation is the second challenge.
- The problem with product development standards while establishing an automated testing procedure in place.
- Evaluation of several automated software technologies will determine the most effective one.

- The problem of both time and money arises because of the large initial expenditure of both while the testing.

ii. Test Tool Selection:

The second stage of the test automation life-cycle approach is test tool selection. This stage assists the person in deciding which testing tool to employ and what method to evaluate it. Since essentially all testing requirements are supported by the testing tool, the inspector still has to study the concurrent engineering environment and other organisational demands before compiling a list of tool assessment criteria. Software testers assess the apparatus through using specified sample benchmarks.

iii. Scope Introduction:

The third step of the automation test life-cycle approach is indicated by this phase. The rapid application area is included in the scope of automate. The following criteria serve as a foundation for determining scope:

- Software application qualities that all software applications have within common.
- The reusable scope of business units and departments is established by automation testing.
- Automation testing examines how much the business components can also be reused.
- An application ought to be technically possible and include characteristics that are particular to the company.
- For cross-browser automation, automation testing provides the capacity to repeat test scenarios.
- This stage establishes the overall testing approach, which must be carefully monitored and updated as needed. Testing abilities of a single teammates and the whole team are contrasted to the necessary specialized skills for a certain software application in order to confirm the availability of skills.

Software Testing Methodologies

- Verification is the process of checking that a software achieves its goal without any bugs. It is the process to ensure whether the product that is developed is right or not. It verifies whether the developed product fulfills the requirements that we have. Verification is static testing.
- Validation is the process of checking whether the software product is up to the mark or in other words product has high level requirements. It is the process of checking the validation of product i.e. it checks what we are developing is the right product. It is validation of actual and expected product. Validation is the dynamic testing.

The difference between Verification and Validation is as following Table 4.1:

Table 4.1: Represented that the Different between Validation and Verification

Validation	Verification
It includes testing and validating the actual product.	It includes checking documents, design, codes and programs.

Validation is the dynamic testing.	Verification is the static testing.
It includes the execution of the code.	It does not include the execution of the code.
Methods used in validation are Black Box Testing, White Box Testing and non-functional testing.	Methods used in verification are reviews, walkthroughs, inspections and desk-checking.
It checks whether the software meets the requirements and expectations of a customer or not.	It checks whether the software conforms to specifications or not.
It can only find the bugs that could not be found by the verification process.	It can find the bugs in the early stage of the development.
The goal of validation is an actual product.	The goal of verification is application and software architecture and specification.
Validation is executed on software code with the help of testing team.	Quality assurance team does verification.
It comes after verification.	It comes before validation.
It consists of execution of program and is performed by computer.	It consists of checking of documents/files and is performed by human.

Grey Box Testing

Gray Box Testing is a software testing technique which is a combination of Black Box Testing technique and White Box Testing technique. In Black Box Testing technique, tester is unknown to the internal structure of the item being tested and in White Box Testing the internal structure is known to tester. The internal structure is partially known in Gray Box Testing. This includes access to internal data structures and algorithms for purpose of designing the test cases.

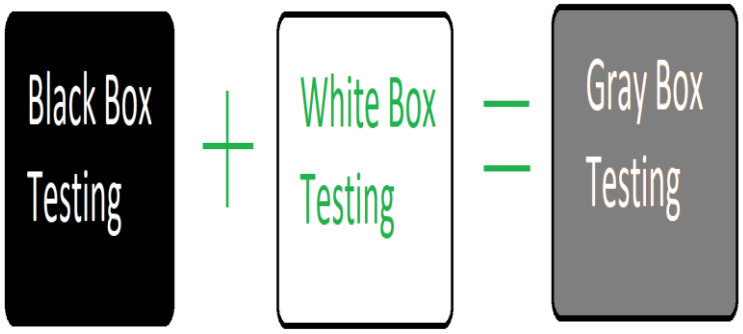


Figure 4.3: Represented that the Block Diagram of Gray Box Testing

Gray Box Testing as display in below Figure 4.3, is named so because the software program is like a semitransparent or grey box inside which tester can partially see. It commonly focuses on context-specific errors related to web systems. It is based on requirement test case generation because it has all the conditions presented before the program is tested.

Objective of Gray Box Testing:

The objective of Gray Box Testing is:

- **a.** To provide combined advantages of both black box testing and white box testing.
- **b.** To combine the input of developers as well as testers.
- **c.** To improve overall product quality.
- **d.** To reduce the overhead of long process of functional and non-functional testings.
- **e.** To provide enough free time to developers to fix defects.
- **f.** To test from the user point of view rather than a designer point of view.

Gray Box Testing Techniques:

Matrix Testing:

In matrix testing technique, business and technical risks which are defined by the developers in software programs are examined. Developers define all the variables that exist in the program. Each of the variables has an inherent technical and business risk and can be used with varied frequencies during its life cycle.

Pattern Testing:

To perform the testing, previous defects are analyzed. It determines the cause of the failure by looking into the code. Analysis template includes reasons for the defect. This helps test cases designed as they are proactive in finding other failures before hitting production.

Orthogonal Array Testing:

It is mainly a black box testing technique. In orthogonal array testing, test data have n numbers of permutations and combinations. Orthogonal array testing is preferred when maximum coverage is required when there are very few test cases and test data is large. This is very helpful in testing complex applications.

Regression Testing:

Regression testing is testing the software after every change in the software to make sure that the changes or the new functionalities are not affecting the existing functioning of the system. Regression testing is also carried out to ensure that fixing any defect has not affected other functionality of the software.

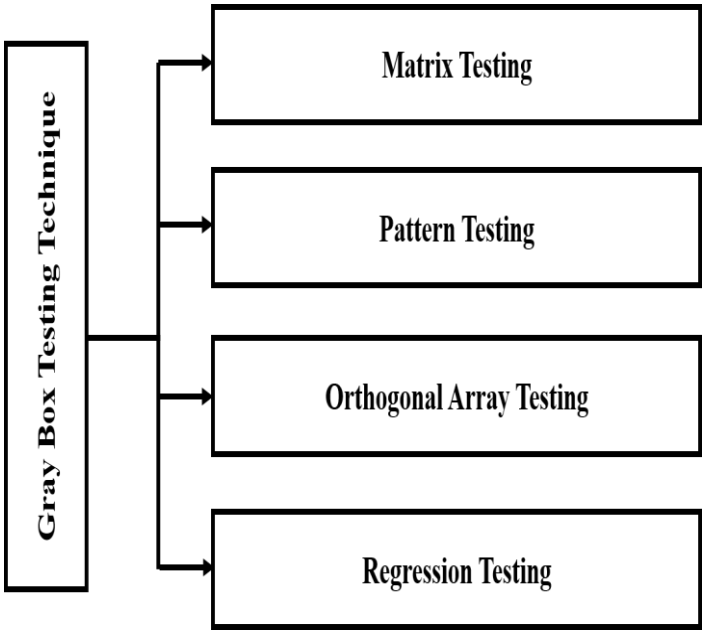


Figure 4.4: Illustrated that the Gray Box testing Techniques.

Advantages of Gray Box Testing:

- Users and developers have clear goals while doing testing.
- Gray box testing is mostly done by the user perspective.
- Testers are not required to have high programming skills for this testing.
- Gray box testing is non-intrusive.
- Overall quality of the product is improved.
- In gray box testing, developers have more time for defect fixing.
- By doing gray box testing, benefits of both black box and white box testing is obtained.
- Gray box testing is unbiased. It avoids conflicts between a tester and a developer.
- Gray box testing is much more effective in integration testing.

Disadvantages of gray box testing:

- Defect association is difficult when gray testing is performed for distributed systems.
- Limited access to internal structure leads to limited access for code path traversal.
- Because source code cannot be accessed, doing complete white box testing is not possible.
- Gray box testing is not suitable for algorithm testing.
- Most of the test cases are difficult to design.

Branch Coverage Testing

Branch coverage technique is used to cover all branches of the control flow graph. It covers all the possible outcomes (true and false) of each condition of decision point at least once. Branch

coverage technique is a white box testing technique that ensures that every branch of each decision point must be executed.

However, branch coverage technique and decision coverage technique are very similar, but there is a key difference between the two.

Decision coverage technique covers all branches of each decision point whereas branch testing covers all branches of every decision point of the code.

In other words, branch coverage follows decision point and branch coverage edges. Many different metrics can be used to find branch coverage and decision coverage, but some of the most basic metrics are: finding the percentage of program and paths of execution during the execution of the program.

Calculation of Branch Coverage

There are several methods to calculate Branch coverage, but path finding is the most common method.

In this method, the number of paths of executed branches is used to calculate Branch coverage.

Branch coverage technique can be used as the alternative of decision coverage. Somewhere, it is not defined as an individual technique, but it is distinct from decision coverage and essential to test all branches of the control flow graph.

Example:

```
Read X
Read Y
IF X+Y > 100 THEN
Print "Large"
ENDIF
If X + Y<100 THEN
Print "Small"
ENDIF
```

This is the basic code structure where we took two variables X and Y and two conditions. If the first condition is true, then print "Large" and if it is false, then go to the next condition. If the second condition is true, then print "Small."

Control Flow Graph of Code Structure

In the above Figure 4.5, control flow graph of code is depicted. In the first case traversing through "Yes "decision, the path is A1-B2-C4-D6-E8, and the number of covered edges is 1, 2, 4, 5, 6 and

8 but edges 3 and 7 are not covered in this path. To cover these edges, we have to traverse through "No" decision. In the case of "No" decision the path is A1-B3-5-D7, and the number of covered edges is 3 and 7. So by traveling through these two paths, all branches have covered.

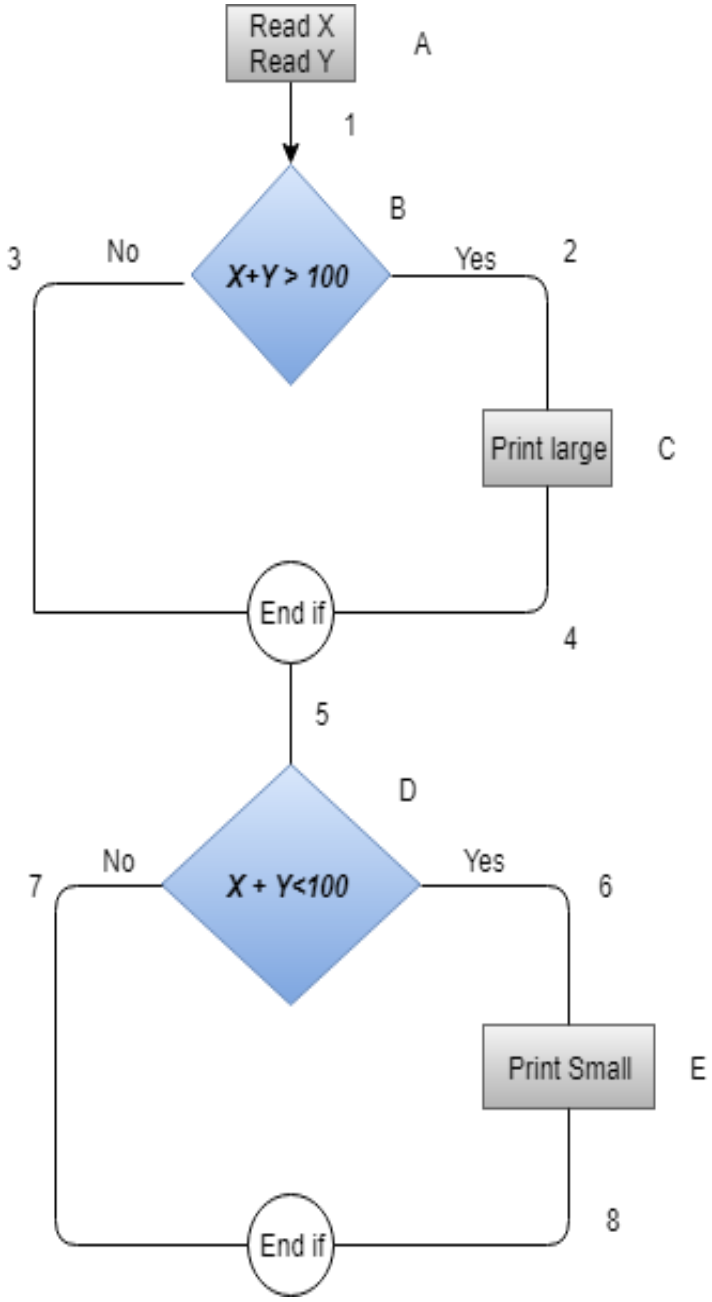


Figure 4.5: Display the Structure of the Code.

Path 1 - A1-B2-C4-D6-E8

Path 2 - A1-B3-5-D7

Branch Coverage (BC) = Number of paths

=2

Path Testing

Path Testing is a method that is used to design the test cases. In path testing method, the control flow graph of a program is designed to find a set of linearly independent paths of execution. In this method Cyclomatic Complexity is used to determine the number of linearly independent paths and then test cases are generated for each path as display in below Figure 4.6.

It give complete branch coverage but achieves that without covering all possible paths of the control flow graph. McCabe's Cyclomatic Complexity is used in path testing. It is a structural testing method that uses the source code of a program to find every possible executable path.

Path Testing Process:

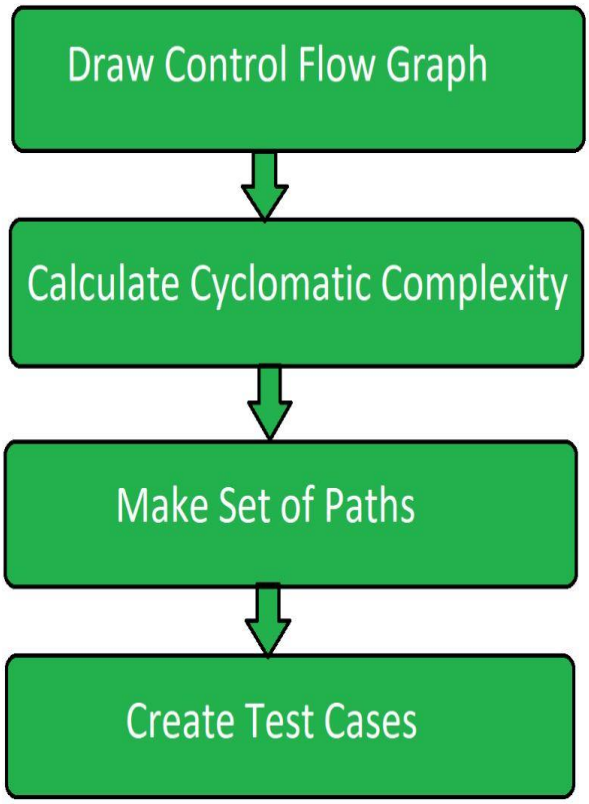


Figure 4.6: Represented that the Path Testing Process.

Path Testing Techniques:

- **Control Flow Graph:**
  The program is converted into control flow graph by representing the code into nodes and edges.
- **Decision to Decision path:**
  The control flow graph can be broken into various Decision to Decision paths and then collapsed into individual nodes.
- **Independent paths:**
  Independent path is a path through a Decision to Decision path graph which cannot be reproduced from other paths by other methods.

Advantages of Path Testing:

- Path testing method reduces the redundant tests.
- Path testing focuses on the logic of the programs.
- Path testing is used in test case design.

Loop Coverage Testing

Loop Testing is a type of software testing type that is performed to validate the loops. It is one of the type of Control Structure Testing. Loop testing is a white box testing technique and is used to test loops in the program.

Objectives of Loop Testing:

The objective of Loop Testing is:

- To fix the infinite loop repetition problem.
- To know the performance.
- To identify the loop initialization problems.
- To determine the uninitialized variables.

Types of Loop Testing:

Loop testing is classified on the basis of the types of the loops:

Simple Loop Testing:

Testing performed in a simple loop is known as Simple loop testing. Simple loop is basically a normal “for”, “while” or “do-while” in which a condition is given and loop runs and terminates according to true and false occurrence of the condition respectively. This type of testing is performed basically to test the condition of the loop whether the condition is sufficient to terminate loop after some point of time.

Example:

```
while(condition)
{
 statement(s);
}
```

Nested Loop Testing:

Testing performed in a nested loop in known as Nested loop testing. Nested loop is basically one loop inside another loop. In nested loop there can be finite number of loops inside a loop and there a nest is made. It may be either of any of three loops i.e., for, while or do-while.

Example:

```
while(condition 1)
 {
```

```
  while(condition 2)
   {
    statement(s);
   }
 }
```

Concatenated Loop Testing:

Testing performed in a concatenated loop is known as Concatenated loop testing. It is performed on the concatenated loops. Concatenated loops are loops after the loop. It is a series of loops. Difference between nested and concatenated is that in nested loop is inside the loop but here loop is after the loop.

Example:

```
while(condition 1)
 {
  statement(s);
 }
while(condition 2)
{
 statement(s);
}
```

Unstructured Loop Testing:

Testing performed in an unstructured loop is known as unstructured loop testing. Unstructured loop is the combination of nested and concatenated loops. It is basically a group of loops that are in no order.

Example:

```
while()
 {
  for()
   {}
  while()
   {}
 }
```

Advantages of Loop Testing:

The advantages of Loop testing are:

- Loop testing limits the number of iterations of loop.
- Loop testing ensures that program doesn't go into infinite loop process.
- Loop testing endures initialization of every used variable inside the loop.
- Loop testing helps in identification of different problems inside the loop.
- Loop testing helps in determination of capacity.

Disadvantages of Loop Testing:

The disadvantages of Loop testing are:

- Loop testing is mostly effective in bug detection in low-level software.
- Loop testing is not useful in bug detection.

Boundary Value Analysis

Functional testing is a type of software testing in which the system is tested against the functional requirements of the system. It is conducted to ensure that the requirements are properly satisfied by the application. Functional testing verifies that each function of the software application works in conformance with the requirement and specification. Boundary Value Analysis (BVA) is one of the functional testing's.

Boundary Value Analysis

Boundary Value Analysis is based on testing the boundary values of valid and invalid partitions. The behavior at the edge of the equivalence partition is more likely to be incorrect than the behavior within the partition, so boundaries are an area where testing is likely to yield defects. It checks for the input values near the boundary that have a higher chance of error. Every partition has its maximum and minimum values and these maximum and minimum values are the boundary values of a partition.

- A boundary value for a valid partition is a valid boundary value.
- A boundary value for an invalid partition is an invalid boundary value.
- For each variable we check-
    - Minimum value.
    - Just above the minimum.
    - Nominal Value.
    - Just below Max value.
    - Max value.

State Transition Testing

State Transition Testing is a type of software testing which is performed to check the change in the state of the application under varying input. The condition of input passed is changed and the change in state is observed as below Figure 4.7.
State Transition Testing is basically a black box testing technique that is carried out to observe the behavior of the system or application for different input conditions passed in a sequence. In this type of testing, both positive and negative input values are provided and the behavior of the

system is observed. State Transition Testing is basically used where different system transitions are needed to be tested.

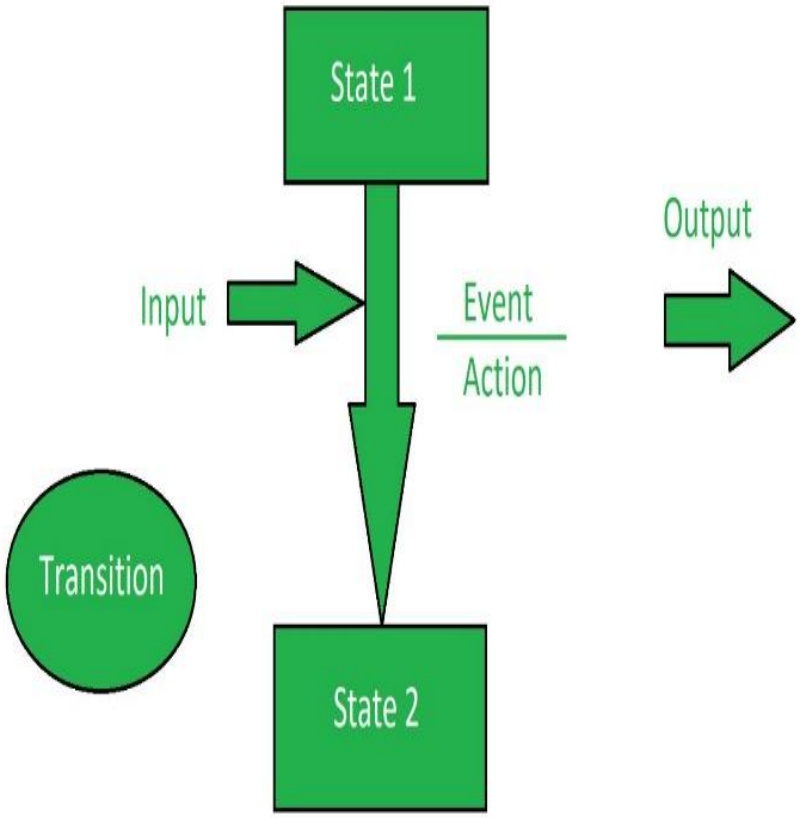


Figure 4.7: Represented that the State Transition Testing.

Objectives of State Transition Testing:

The objective of State Transition testing is:

- To test the behavior of the system under varying input.
- To test the dependency on the values in the past.
- To test the change in transition state of the application.
- To test the performance of the system.

Transition States:

- **Change Mode:**

When this mode is activated then the display mode moves from TIME to DATE.

- **Reset:**

When the display mode is TIME or DATE, then reset mode sets them to ALTER TIME or ALTER DATE respectively.

- **Time Set:**

When this mode is activated, display mode changes from ALTER TIME to TIME.

- **Date Set:**

When this mode is activated, display mode changes from ALTER DATE to DATE.

State Transition Diagram:

State Transition Diagram shows how the state of the system changes on certain inputs. It has four main components:

- States
- Transition
- Events
- Actions

Advantages of State Transition Testing:

- State transition testing helps in understanding the behavior of the system.
- State transition testing gives the proper representation of the system behavior.
- State transition testing covers all the conditions.

Disadvantages of State Transition Testing:

- State transition testing cannot be performed everywhere.
- State transition testing is not always reliable.

Cause Effect Graphing

Cause Effect Graphing based technique is a technique in which a graph is used to represent the situations of combinations of input conditions.

The graph is then converted to a decision table to obtain the test cases. Cause-effect graphing technique is used because boundary value analysis and equivalence class partitioning methods do not consider the combinations of input conditions.

But since there may be some critical behaviours to be tested when some combinations of input conditions are considered, that is why cause-effect graphing technique is used.

Steps used in deriving test cases using this technique are:

- **Division of specification:**

Since it is difficult to work with cause-effect graphs of large specifications as they are complex, the specifications are divided into small workable pieces and then converted into cause-effect graphs separately.

- **Identification of cause and effects:**

This involves identifying the causes (distinct input conditions) and effects (output conditions) in the specification.

- **Transforming the specifications into a cause-effect graph:**

The causes and effects are linked together using Boolean expressions to obtain a cause-effect graph. Constraints are also added between causes and effects if possible.

- **Conversion into decision table:**

The cause-effect graph is then converted into a limited entry decision table. If you're not aware of the concept of decision tables, check out this link.

- **Deriving test cases:**

Each column of the decision-table is converted into a test case.

Decision Table Testing

Decision table testing is a software testing technique used to test system behavior for different input combinations. This is a systematic approach where the different input combinations and their corresponding system behavior (Output) are captured in a tabular form. That is why it is also called as a Cause-Effect table where Cause and effects are captured for better test coverage.

A Decision Table is a tabular representation of inputs versus rules/cases/test conditions. It is a very effective tool used for both complex software testing and requirements management. A decision table helps to check all possible combinations of conditions for testing and testers can also identify missed conditions easily. The conditions are indicated as True (T) and False (F) values.

Parts of Decision Tables:

In software testing, the decision table has 4 parts which are divided into portions and are given below Figure 4.8:

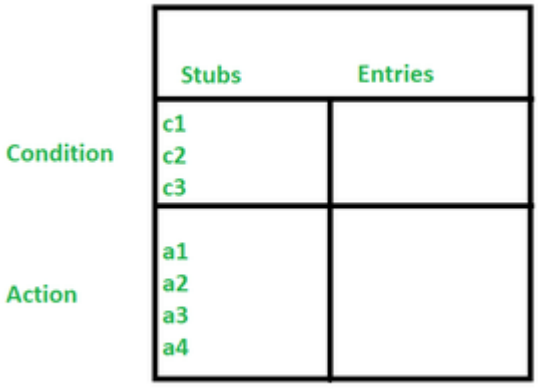


Figure 4.8: Represented that the Parts of Decision Tables.

- **Condition Stubs:**

The conditions are listed in this first upper left part of the decision table that is used to determine a particular action or set of actions.

- **Action Stubs:**

All the possible actions are given in the first lower left portion (i.e, below condition stub) of the decision table.

- **Condition Entries:**

In the condition entry, the values are inputted in the upper right portion of the decision table. In the condition entries part of the table, there are multiple rows and columns which are known as Rule.

- **Action Entries:**

In the action entry, every entry has some associated action or set of actions in the lower right portion of the decision table and these values are called outputs.

Types of Decision Tables:

The decision tables are categorized into two types and these are given below:

- **Limited Entry:**

In the limited entry decision tables, the condition entries are restricted to binary values.

- **Extended Entry:**

In the extended entry decision table, the condition entries have more than two values. The decision tables use multiple conditions where a condition may have many possibilities instead of only 'true' and 'false' are known as extended entry decision tables.

Applicability of Decision Tables:

- The order of rule evaluation has no effect on the resulting action.
- The decision tables can be applied easily at the unit level only.
- Once a rule is satisfied and the action selected, n another rule needs to be examined.
- The restrictions do not eliminate many applications.

Use Case Testing

Use Case Testing is a software testing technique that helps to identify test cases that cover entire system on a transaction by transaction basis from start to end. Test cases are the interactions between users and software application.

Use case testing helps to identify gaps in software application that might not be found by testing individual software components. A Use Case in Testing is a brief description of a particular use of the software application by an actor or user. Use cases are made on the basis of user actions and the response of the software application to those user actions. It is widely used in developing test cases at system or acceptance level.

Feature of Use Case Testing:

There is some feature of a use case testing, which is used to test the software project and provide a better response, these are given below:

- Use case testing is not testing that is performed to decide the quality of the software.
- Although it is a type of end to end testing, it won't ensure the entire coverage of the user application.
- Use Cases has generally captured the interactions between 'actors' and the 'system'.
- 'Actors' represents the user and their interactions that each user takes part in.
- The use case will find out the defects in integration testing.

Benefits of Use Case Testing:

Use case testing provide some functionality which is used to help to develop a software project. These are given below:

- Use case driven analysis is that it helps manage complexity since it focuses on one specific usage aspect at a time.
- Use cases start from a very simple view that a system is built first and foremost for its users.
- Use cases are a sequence of steps that describe the interactions between the actor and the system.
- Use case help to capture the functional requirements of a system.
- Use cases are used to hearten designers to outcomes before attempting to specify outcomes, and thereby they help to make requirements more proactive in system development.

Exploratory Testing

Exploratory Testing is a type of software testing where Test cases are not created in advance but testers check system on the fly.

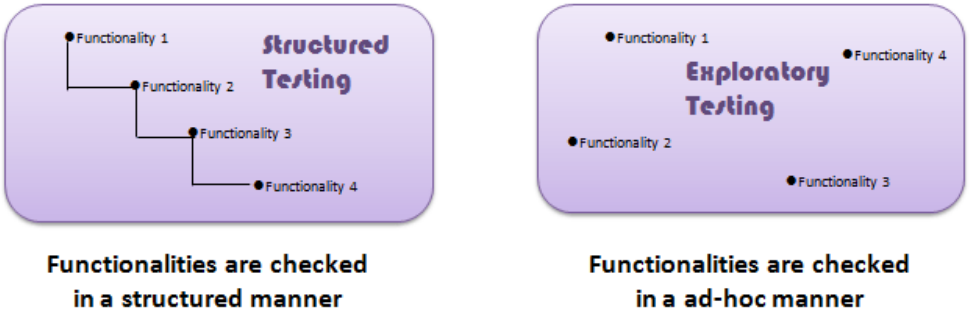
They may note down ideas about what to test before test execution.

The focus of exploratory testing is more on testing as a "thinking" activity. Exploratory Testing is widely used in Agile models and is all about discovery, investigation, and learning. It emphasizes personal freedom and responsibility of the individual tester.

Need of Exploratory Testing

Under scripted testing, you design test cases first and later proceed with test execution. On the contrary, exploratory testing is a simultaneous process of test design and test execution all done at the same time as display in below Figure 4.9.

Scripted Test Execution is usually a non-thinking activity where testers execute the test steps and compare the actual results with expected results. Such test execution activity can be automated does not require many cognitive skills.



Functionalities are checked in a structured manner

Functionalities are checked in a ad-hoc manner

Figure 4.9: Represented that the Exploratory Testing.

Exploratory Testing Techniques

- Is not random testing but it is ad-hoc testing with a purpose of find bugs
- Is structured and rigorous
- Is cognitively (thinking) structured as compared to the procedural structure of scripted testing. This structure comes from Charter, time boxing etc.
- Is highly teachable and manageable
- It is not a technique but it is an approach. What actions you perform next is governed by what you are doing currently

Pros and Cons of Exploratory Testing

Advantages

- This testing is useful when requirement documents are not available or partially available
- It involves Investigation process which helps find more bugs than normal testing-
- Uncover bugs which are normally ignored by other testing techniques
- Helps to expand the imagination of testers by executing more and more test cases which finally improves productivity as well
- This testing drill down to the smallest part of an application and covers all the requirements
- This testing covers all the types of testing and it covers various scenarios and cases
- Encourages creativity and intuition
- Generation of new ideas during test execution

Disadvantages

- This testing purely depends on the tester skills
- Limited by domain knowledge of the tester
- Not suitable for Long execution time

Challenges of Exploratory Testing:

There are many challenges of exploratory testing and those are explained below:

- Learning to use the application or software system is a challenge
- Replication of failure is difficult
- Determining whether tools need to be used can be challenging
- Determine the best test cases to execute can be difficult
- Reporting of the test results is a challenge as the report doesn't have planned scripts or cases to compare with the actual result or outcome
- Documentation of all events during execution is difficult to record
- Don't know when to stop the testing as exploratory testing has definite test cases to execute.

Software Testing Life Cycle:

Requirements Analysis/Design

Requirement analysis is significant and essential activity after elicitation. We analyze, refine, and scrutinize the gathered requirements to make consistent and unambiguous requirements. This activity reviews all requirements and may provide a graphical view of the entire system. After the completion of the analysis, it is expected that the understandability of the project may improve significantly. Here, we may also use the interaction with the customer to clarify points of confusion and to understand which requirements are more important than others.

The various steps of requirement analysis are shown in below Figure 4.10:

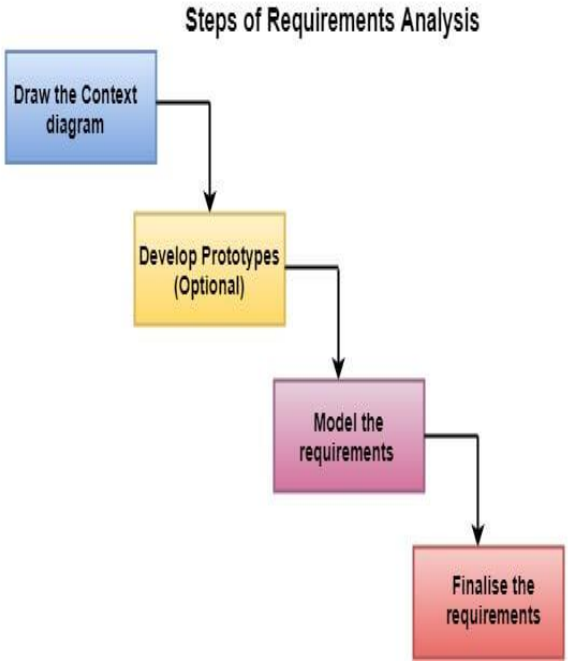


Figure 4.10: Illustrated that the Requirements and Analysis Steps.

i. Draw the Context Diagram:

The context diagram is a simple model that defines the boundaries and interfaces of the proposed systems with the external world. It identifies the entities outside the proposed system that interact with the system. The context diagram of student result management system is given below Figure 4.11:

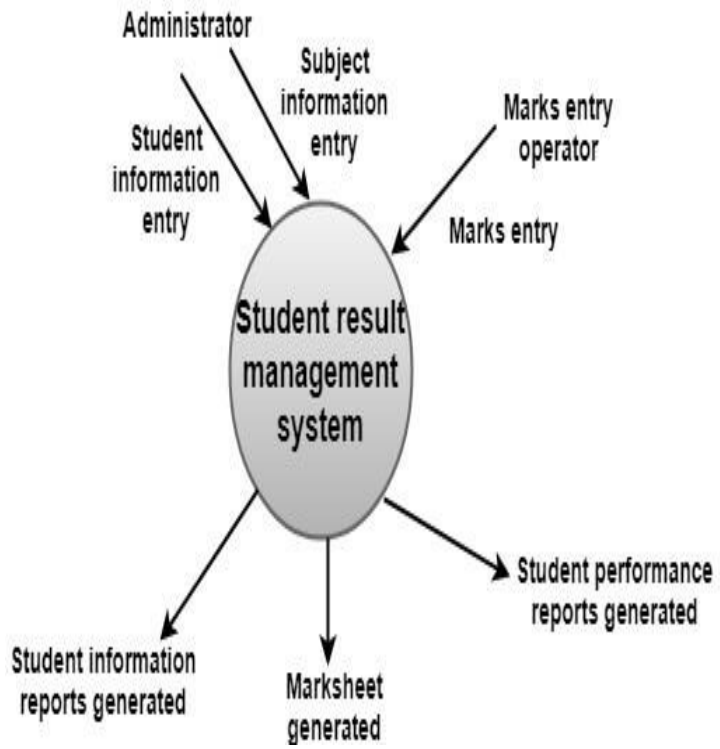


Figure 4.11: Showed that the Context diagram of Management System.

ii. Development of a Prototype (optional):

One effective way to find out what the customer wants is to construct a prototype, something that looks and preferably acts as part of the system they say they want.

We can use their feedback to modify the prototype until the customer is satisfied continuously. Hence, the prototype helps the client to visualize the proposed system and increase the understanding of the requirements. When developers and users are not sure about some of the elements, a prototype may help both the parties to take a final decision.

Some projects are developed for the general market. In such cases, the prototype should be shown to some representative sample of the population of potential purchasers. Even though a person who tries out a prototype may not buy the final system, but their feedback may allow us to make the product more attractive to others.

The prototype should be built quickly and at a relatively low cost. Hence it will always have limitations and would not be acceptable in the final system. This is an optional activity.

iii. Model the Requirements:

This process usually consists of various graphical representations of the functions, data entities, external entities, and the relationships between them. The graphical view may help to find

incorrect, inconsistent, missing, and superfluous requirements. Such models include the Data Flow diagram, Entity-Relationship diagram, Data Dictionaries, State-transition diagrams, etc.

iv. Finalize the Requirements:

After modeling the requirements, we will have a better understanding of the system behavior. The inconsistencies and ambiguities have been identified and corrected. The flow of data amongst various modules has been analyzed. Elicitation and analyze activities have provided better insight into the system. Now we finalize the analyzed requirements, and the next step is to document these requirements in a prescribed format.

CHAPTER 5

AN INTRODUCTION OF TRACEABILITY MATRIX

Jayaprakash B, Assistant Professor
Department of Computer Science & IT, School of Sciences, Jain (Deemed-to-be University), Bangalore-27, India
Email Id- b.jayaprakash@jainuniversity.ac.in

Traceability matrix is a table type document that is used in the development of software application to trace requirements. It can be used for both forward (from Requirements to Design or Coding) and backward (from Coding to Requirements) tracing. It is also known as Requirement Traceability Matrix (RTM) or Cross Reference Matrix (CRM). It is prepared before the test execution process to make sure that every requirement is covered in the form of a Test case so that we don't miss out any testing. In the RTM document, we map all the requirements and corresponding test cases to ensure that we have written all the test cases for each condition. The test engineer will prepare RTM for their respective assign modules, and then it will be sent to the Test Lead. The Test Lead will go repository to check whether the Test Case is there or not and finally Test Lead consolidate and prepare one necessary RTM document.

This document is designed to make sure that each requirement has a test case, and the test case is written based on business needs, which are given by the client. It will be performed with the help of the test cases if any requirement is missing, which means that the test case is not written for a particular need, and that specific requirement is not tested because it may have some bugs. The traceability is written to make sure that the entire requirement is covered.

We can observe in the below image that the requirement number 2 and 4 test case names are not mentioned that's why we highlighted them, so that we can easily understand that we have to write the test case for them.

Generally, this is like a worksheet document, which contains a table, but there are also many user-defined templates for the traceability matrix. Each requirement in the traceability matrix is connected with its respective test case so that tests can be carried out sequentially according to specific requirements.

Goals of Traceability Matrix

- It helps in tracing the documents that are developed during various phases of SDLC.
- It ensures that the software completely meets the customer's requirements.
- It helps in detecting the root cause of any bug.

Types of Traceability Test Matrix

The traceability matrix can be classified into three different types which are as follows:

- Forward traceability

- Backward or reverse traceability
- Bi-directional traceability

Forward Traceability

The forward traceability test matrix is used to ensure that every business's needs or requirements are executed correctly in the application and also tested rigorously as mentioned in below Figure 5.1. The main objective of this is to verify whether the product developments are going in the right direction. In this, the requirements are mapped into the forward direction to the test cases.

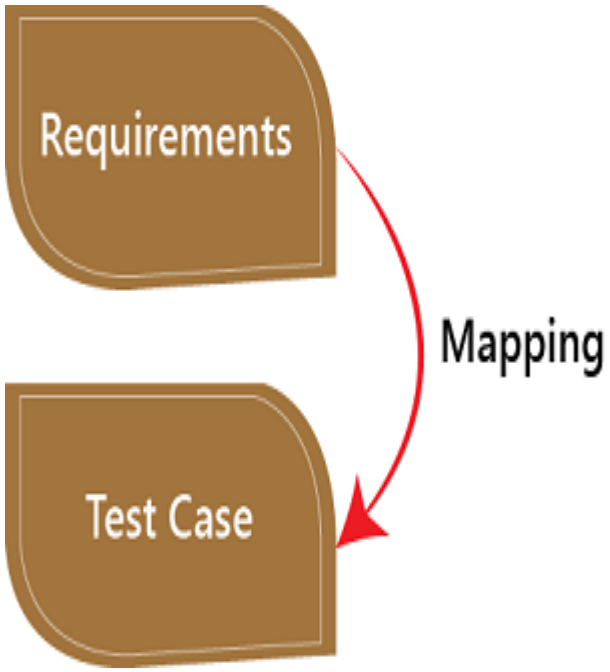


Figure 5.1: Represented that the Mapping Forward Traceability.

Backward or Reverse Traceability

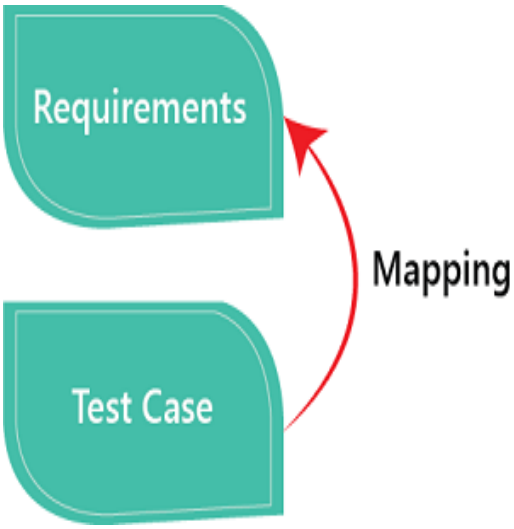


Figure 5.2: Represented that the Backward or Reverse Traceability.

The reverse or backward traceability is used to check that we are not increasing the space of the product by enhancing the design elements, code, test other things which are not mentioned in the business needs. And the main objective of this that the existing project remains in the correct direction. In this, the requirements are mapped into the backward direction to the test cases as display in below Figure 5.2.

Bi-directional Traceability

It is a combination of forwarding and backward traceability matrix, which is used to make sure that all the business needs are executed in the test cases as in Figure 5.3. It also evaluates the modification in the requirement which is occurring due to the bugs in the application.

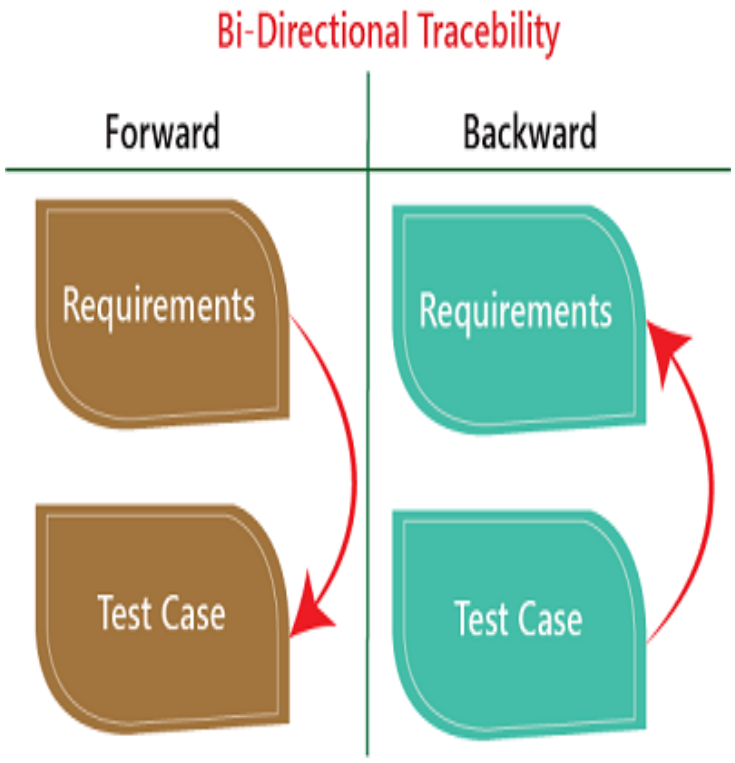


Figure 5.3: Represented that the Bi-directional Traceability.

Advantage of RTM

Following are the benefits of requirement traceability matrix:

- With the help of the RTM document, we can display the complete test execution and bugs status based on requirements.
- It is used to show the missing requirements or conflicts in documents.
- In this, we can ensure the complete test coverage, which means all the modules are tested.
- It will also consider the efforts of the testing teamwork towards reworking or reconsidering on the test cases.

Test Planning

A test plan is a detailed document which describes software testing areas and activities. It outlines the test strategy, objectives, test schedule, required resources (human resources, software, and hardware), test estimation and test deliverables. The test plan is a base of every software's testing. It is the most crucial activity which ensures availability of all the lists of planned activities in an appropriate sequence. The test plan is a template for conducting software testing activities as a defined process that is fully monitored and controlled by the testing manager. The test plan is prepared by the Test Lead (60%), Test Manager (20%), and by the test engineer (20%).

Types of Test Plan

There are three types of the test plan:

- Master Test Plan
- Phase Test Plan
- Testing Type Specific Test Plans

i. Master Test Plan

Master Test Plan is a type of test plan that has multiple levels of testing. It includes a complete test strategy.

ii. Phase Test Plan

A phase test plan is a type of test plan that addresses any one phase of the testing strategy. For example, a list of tools, a list of test cases, etc.

iii. Specific Test Plans

Specific test plan designed for major types of testing like security testing, load testing, performance testing, etc. In other words, a specific test plan designed for non-functional testing.

Write a Test Plan

Making a test plan is the most crucial task of the test management process. According to IEEE 829, follow the following seven steps to prepare a test plan.

- First, analyze product structure and architecture.
- Now design the test strategy.
- Define all the test objectives.
- Define the testing area.
- Define all the useable resources.
- Schedule all activities in an appropriate manner.
- Determine all the Test Deliverables.

Test Plan Guidelines

- Collapse your test plan.
- Avoid overlapping and redundancy.
- If you think that you do not need a section that is already mentioned above, then delete that section and proceed ahead.
- Be specific. For example, when you specify a software system as the part of the test environment, then mention the software version instead of only name.
- Avoid lengthy paragraphs.
- Use lists and tables wherever possible.
- Update plan when needed.
- Do not use an outdated and unused document.

Importance of Test Plan

- The test plan gives direction to our thinking. This is like a rule book, which must be followed.
- The test plan helps in determining the necessary efforts to validate the quality of the software application under the test.
- The test plan helps those people to understand the test details that are related to the outside like developers, business managers, customers, etc.
- Important aspects like test schedule, test strategy, test scope etc are documented in the test plan so that the management team can review them and reuse them for other similar projects.

Test Cases Design

Write Test Cases

A Test Case is a set of actions executed to verify a particular feature or functionality of your software application. A Test Case contains test steps, test data, precondition, postcondition developed for specific test scenario to verify any requirement. The test case includes specific variables or conditions, using which a testing engineer can compare expected and actual results to determine whether a software product is functioning as per the requirements of the customer.

Best Practice for writing good Test Case

i. Test Cases need to be simple and transparent:

Create test cases that are as simple as possible. They must be clear and concise as the author of the test case may not execute them.

Use assertive language like go to the home page, enter data, click on this and so on. This makes the understanding the test steps easy and tests execution faster.

ii. Create Test Case with End User in Mind

The ultimate goal of any software project is to create test cases that meet customer requirements and is easy to use and operate. A tester must create test cases keeping in mind the end user perspective

iii. Avoid test case repetition.

Do not repeat test cases. If a test case is needed for executing some other test case, call the test case by its test case id in the pre-condition column

iv. Do not Assume

Do not assume functionality and features of your software application while preparing test case. Stick to the Specification Documents.

v. Ensure 100% Coverage

Make sure you write test cases to check all software requirements mentioned in the specification document. Use Traceability Matrix to ensure no functions/conditions is left untested.

vi. Test Cases must be identifiable.

Name the test case id such that they are identified easily while tracking defects or identifying a software requirement at a later stage.

vii. Implement Testing Techniques

It's not possible to check every possible condition in your software application. Software Testing techniques help you select a few test cases with the maximum possibility of finding a defect.

- **Boundary Value Analysis (BVA):** As the name suggests it's the technique that defines the testing of boundaries for a specified range of values.
- **Equivalence Partition (EP):** This technique partitions the range into equal parts/groups that tend to have the same behavior.
- **State Transition Technique**: This method is used when software behavior changes from one state to another following particular action.
- **Error Guessing Technique:** This is guessing/anticipating the error that may arise while doing manual testing. This is not a formal method and takes advantages of a tester's experience with the application

viii. Self-cleaning

The test case you create must return the Test Environment to the pre-test state and should not render the test environment unusable. This is especially true for configuration testing.

ix. Repeatable and self-standing

The test case should generate the same results every time no matter who tests it

x. Peer Review.

After creating test cases, get them reviewed by your colleagues. Your peers can uncover defects in your test case design, which you may easily miss.

While drafting a test case to include the following information

- The description of what requirement is being tested
- The explanation of how the system will be tested.
- The test setup like a version of an application under test, software, data files, operating system, hardware, security access, physical or logical date, time of day, prerequisites such as other tests and any other setup information pertinent to the requirements being tested.
- Inputs and outputs or actions and expected results.
- Any proofs or attachments.
- Use active case language.
- Test Case should not be more than 15 steps.
- An automated test script is commented with inputs, purpose and expected results.
- The setup offers an alternative to pre-requisite tests.
- With other tests, it should be an incorrect business scenario order.

Test Case Management Tools

Test management tools are the automation tools that help to manage and maintain the Test Cases. Main Features of a test case management tool are

i. **For documenting Test Cases:**

With tools, you can expedite Test Case creation with use of templates

ii. **Execute the Test Case and Record the results:**

Test Case can be executed through the tools and results obtained can be easily recorded.

iii. **Automate the Defect Tracking:**

Failed tests are automatically linked to the bug tracker, which in turn can be assigned to the developers and can be tracked by email notifications.

iv. **Traceability:**

Requirements, Test cases, Execution of Test cases are all interlinked through the tools, and each case can be traced to each other to check test coverage.

v. **Protecting Test Cases:**

Test cases should be reusable and should be protected from being lost or corrupted due to poor version control. Test Case Management Tools offer features like

- Naming and numbering conventions
- Versioning
- Read-only storage
- Controlled access
- Off-site backup

Types of Test Cases

The test case is defined as a group of conditions under which a tester determines whether a software application is working as per the customer's requirements or not. Test case designing includes preconditions, case name, input conditions, and expected result. A test case is a first level action and derived from test scenarios as display in Figure 5.4.

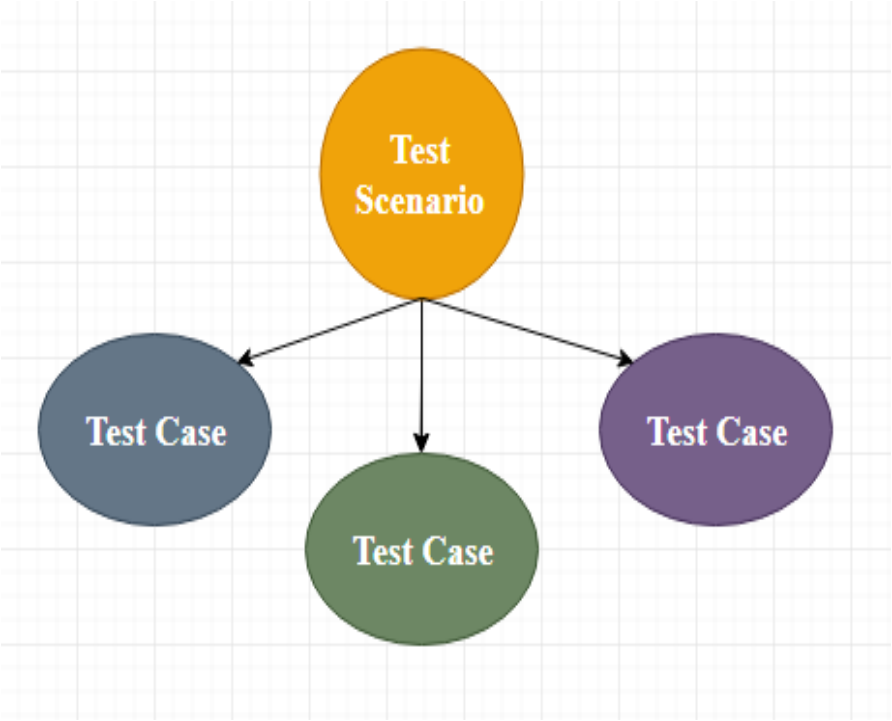


Figure 5.4: Represented that the Different Test Cases.

It is an in-details document that contains all possible inputs (positive as well as negative) and the navigation steps, which are used for the test execution process. Writing of test cases is a one-time attempt that can be used in the future at the time of regression testing. Test case gives detailed information about testing strategy, testing process, preconditions, and expected output. These are executed during the testing process to check whether the software application is performing the task for that it was developed or not. Test case helps the tester in defect reporting by linking defect with test case ID. Detailed test case documentation works as a full proof guard for the testing team because if developer missed something, then it can be caught during execution of these full-proof test cases. To write the test case, we must have the requirements to derive the inputs, and the test scenarios must be written so that we do not miss out on any features for testing. Then we should have the test case template to maintain the uniformity, or every test engineer follows the same approach to prepare the test document.

Generally, we will write the test case whenever the developer is busy in writing the code.

Timing of the Write Test Case

We will write the test case when we get the following:

- When the customer gives the business needs then, the developer starts developing and says that they need 3.5 months to build this product.

- And In the meantime, the testing team will start writing the test cases.
- Once it is done, it will send it to the Test Lead for the review process.
- And when the developers finish developing the product, it is handed over to the testing team.
- The test engineers never look at the requirement while testing the product document because testing is constant and does not depends on the mood of the person rather than the quality of the test engineer.

Need for write the Test Cases

We will write the test for the following reasons:

- To require consistency in the test case execution
- To make sure a better test coverage
- It depends on the process rather than on a person
- To avoid training for every new test engineer on the product
- **To require consistency in the test case execution:** we will see the test case and start testing the application.
- **To make sure a better test coverage:** for this, we should cover all possible scenarios and document it, so that we need not remember all the scenarios again and again.
- **It depends on the process rather than on a person:** A test engineer has tested an application during the first release, second release, and left the company at the time of third release. As the test engineer understood a module and tested the application thoroughly by deriving many values. If the person is not there for the third release, it becomes difficult for the new person. Hence all the derived values are documented so that it can be used in the future.
- **To avoid giving training for every new test engineer on the product:** When the test engineer leaves, he/she leaves with a lot of knowledge and scenarios. Those scenarios should be documented so that the new test engineer can test with the given scenarios and also can write the new scenarios.

Difference between Test Scenarios and Test Cases

In the Table 5.1, that is below we have detailed a few of the important distinctions between test cases and test scenarios:

Table 5.1: Represented that the Difference between Test Scenarios and Test Cases

Test Environment Test Setup

Test Scenarios	Test Cases
The test Scenarios is just a document that is detailed and provides details about the	The test case is just a document that is detailed which provides details about the assessment

assessment method, testing process, preconditions, and anticipated output. The test scenarios are the ones based on the use situation and give one-line information ona what to check.	method, testing process, preconditions, and anticipated output.
These are high-level actions.	These are low-level actions.
It will take less time as compared to test cases.	It takes more time in comparison to try circumstances.
The test scenario will help us in a way that is nimble of through the functionality.	The test case enable our evaluation that is detailed of application.
A lot fewer sources are sufficient to publish test circumstances in comparison with the test instances.	To write the test, we need extra sources to generate and do test situations.
The test Scenarios tend to be work on the essential to “things to be tested”.	The test case is work on the fundamentals of “just how to be tested”.
Test scenarios are really easy to maintain due to their high-level design.	The test cases are hard to preserve.
Writing the test scenario’s primary objective is an address end to get rid of functionality of a software program.	The aim that is main regarding the test case is to verify the test situation by applying steps.
Test scenarios are one-liner statement, however,the it is linked to a few test instances.	It includes all the positive and inputs being negative navigation measures, anticipated results, pre and post condition, etc.

A testing environment is a setup of software and hardware for the testing teams to execute test cases. In other words, it supports test execution with hardware, software and network configured. Test bed or test environment is configured as per the need of the Application under Test. On a few occasion, test bed could be the combination of the test environment and the test data it operates. Setting up a right test environment ensures software testing success. Any flaws in this process may lead to extra cost and time to the client.

Key areas to set up in Test Environment

For the test environment, a key area to set up includes

- System and applications
- Test data

- Database server
- Front-end running environment
- Client operating system
- Browser
- Hardware includes Server Operating system
- Network
- Documentation required like reference documents/configuration guides/installation guides/ user manuals

Process of Software Test Environment Setup

The test environment requires setting up of various number of distinct areas like,

i. Setup of Test Server

Every test may not be executed on a local machine. It may need establishing a test server, which can support applications.

For example, Fedora set up for PHP, Java-based applications with or without mail servers, cron set up, Java-based applications, etc.

ii. Network

Network set up as per the test requirement. It includes,

- Internet setup
- LAN Wifi setup
- Private network setup

It ensures that the congestion that occurs during testing doesn't affect other members. (Developers, designers, content writers, etc.)

iii. Test PC setup

For web testing, you may need to set up different browsers for different testers. For desktop applications, you need various types of OS for different testers PCs.

For example, windows phone app testing may require

- Visual Studio installation
- Windows phone emulator
- Alternatively, assigning a windows phone to the tester.

iv. Bug Reporting

Bug reporting tools should be provided to testers.

Challenges in setting up Test Environment Management

- **Proper Planning on Resource Usage:** Ineffective planning for resource usage can affect the actual output. Also, it may lead to conflict between teams.

- **Remote Environment:** It is possible that a Test environment is located geographically apart. In such a case, the testing team has to rely on the support team for various test assets. (Software, hardware, and other issues).
- **Elaborate Setup Time:** Sometimes test set up gets too elaborated in cases of Integration Testing.
- **Shared Usage by Teams:** If the testing environment is used by development & testing team simultaneously, test results will be corrupted.
- **Complex Test Configuration:** Certain test requires complex test environment configuration. It may pose a challenge to the test team.

Software Requirement Specifications

The production of the requirements stage of the software development process is Software Requirements Specifications (SRS) (also called a requirements document). This report lays a foundation for software engineering activities and is constructing when entire requirements are elicited and analyzed. SRS is a formal report, which acts as a representation of software that enables the customers to review whether it (SRS) is according to their requirements. Also, it comprises user requirements for a system as well as detailed specifications of the system requirements as given in below figure.

The SRS is a specification for a specific software product, program, or set of applications that perform particular functions in a specific environment. It serves several goals depending on who is writing it. First, the SRS could be written by the client of a system. Second, the SRS could be written by a developer of the system. The two methods create entirely various situations and establish different purposes for the document altogether. The first case, SRS, is used to define the needs and expectation of the users. The second case, SRS, is written for various purposes and serves as a contract document between customer and developer.

Characteristics of good SRS

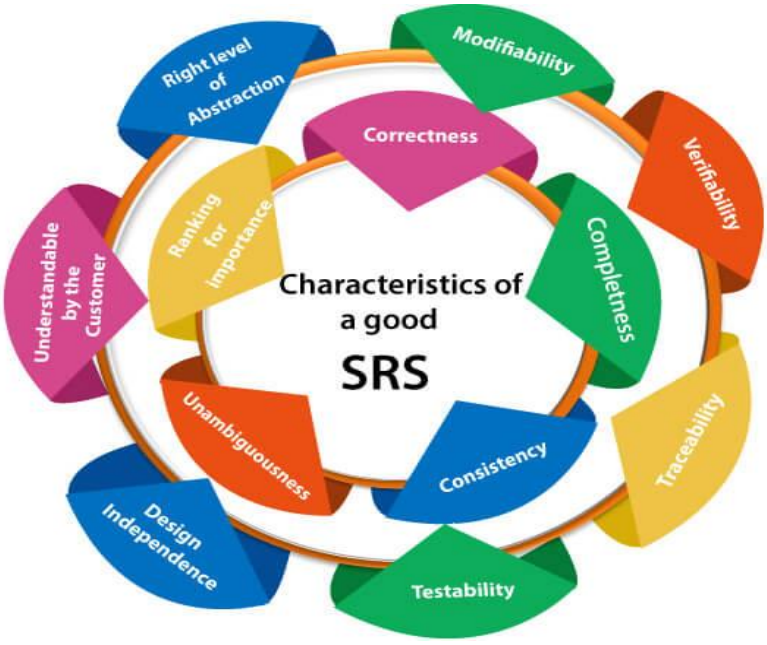


Figure 5.5: Represented that the Characteristics of Good SRS.

Following are the features of a good SRS document:

i. **Correctness:** User review is used to provide the accuracy of requirements stated in the SRS. SRS is said to be perfect if it covers all the needs that are truly expected from the system.

ii. **Completeness:** The SRS is complete if, and only if, it includes the following elements:

- All essential requirements, whether relating to functionality, performance, design, constraints, attributes, or external interfaces.
- Definition of their responses of the software to all realizable classes of input data in all available categories of situations.
- Full labels and references to all figures, tables, and diagrams in the SRS and definitions of all terms and units of measure.

iii. **Consistency:** The SRS is consistent if, and only if, no subset of individual requirements described in its conflict. There are three types of possible conflict in the SRS:

- The specified characteristics of real-world objects may conflicts. For example,
  a. The format of an output report may be described in one requirement as tabular but in another as textual.
  b. One condition may state that all lights shall be green while another states that all lights shall be blue.

iv. There may be a reasonable or temporal conflict between the two specified actions. For example,

  a. One requirement may determine that the program will add two inputs, and another may determine that the program will multiply them.
  b. One condition may state that "A" must always follow "B," while other requires that "A and B" co-occurs.

v. Two or more requirements may define the same real-world object but use different terms for that object. For example, a program's request for user input may be called a "prompt" in one requirement's and a "cue" in another. The use of standard terminology and descriptions promotes consistency.

vi. **Unambiguousness:**

SRS is unambiguous when every fixed requirement has only one interpretation. This suggests that each element is uniquely interpreted. In case there is a method used with multiple definitions, the requirements report should determine the implications in the SRS so that it is clear and simple to understand.

vii. **Ranking for importance and stability:**

The SRS is ranked for importance and stability if each requirement in it has an identifier to indicate either the significance or stability of that particular requirement.

viii. Typically, all requirements are not equally important. Some prerequisites may be essential, especially for life-critical applications, while others may be desirable. Each

element should be identified to make these differences clear and explicit. Another way to rank requirements is to distinguish classes of items as essential, conditional, and optional.

ix. Modifiability:

SRS should be made as modifiable as likely and should be capable of quickly obtain changes to the system to some extent. Modifications should be perfectly indexed and cross-referenced.

x. Verifiability:

SRS is correct when the specified requirements can be verified with a cost-effective system to check whether the final software meets those requirements. The requirements are verified with the help of reviews.

xi. Traceability:

The SRS is traceable if the origin of each of the requirements is clear and if it facilitates the referencing of each condition in future development or enhancement documentation.

Properties of a good SRS Document

The essential properties of a good SRS document are the following:

i. Concise:

The SRS report should be concise and at the same time, unambiguous, consistent, and complete. Verbose and irrelevant descriptions decrease readability and also increase error possibilities.

ii. Structured:

It should be well-structured. A well-structured document is simple to understand and modify. In practice, the SRS document undergoes several revisions to cope up with the user requirements. Often, user requirements evolve over a period of time. Therefore, to make the modifications to the SRS document easy, it is vital to make the report well-structured.

iii. Black-box view:

It should only define what the system should do and refrain from stating how to do these. This means that the SRS document should define the external behavior of the system and not discuss the implementation issues. The SRS report should view the system to be developed as a black box and should define the externally visible behavior of the system. For this reason, the SRS report is also known as the black-box specification of a system.

iv. Conceptual integrity:

It should show conceptual integrity so that the reader can merely understand it. Response to undesired events: It should characterize acceptable responses to unwanted events. These are called system response to exceptional conditions.

v. Verifiable:

All requirements of the system, as documented in the SRS document, should be correct. This means that it should be possible to decide whether or not requirements have been met in an implementation.

Execute Test Cases

The term Test Execution tells that the testing for the product or application needs to be executed in order to obtain the expected result. After the development phase, the testing phase will take place where the various levels of testing techniques will be carried out and the creation and execution of test cases will be taken place. The article focuses on discussing test execution.

Test Execution?

Test Execution is the process of executing the tests written by the tester to check whether the developed code or functions or modules are providing the expected result as per the client requirement or business requirement. Test Execution comes under one of the phases of the Software Testing Life Cycle (STLC).

In the test execution process, the tester will usually write or execute a certain number of test cases, and test scripts or do automated testing. If it creates any errors then it will be informed to the respective development team to correct the issues in the code. If the text execution process shows successful results then it will be ready for the deployment phase after the proper setup for the deployment environment.

Importance of Test Execution:

- **The project runs efficiently:** Test execution ensures that the project runs smoothly and efficiently.
- **Application competency:** It also helps to make sure the application's competency in the global market.
- **Requirements are correctly collected:** Test executions make sure that the requirements are collected correctly and incorporated correctly in design and architecture.
- **Application built in accordance with requirements:** It also checks whether the software application is built in accordance with the requirements or not.

Activities for Test Execution

The following are the 5 main activities that should be carried out during the test execution.

i. **Defect Finding and Reporting:**

Defect finding is the process of identifying the bugs or errors raised while executing the test cases on the developed code or modules. If any error appears or any of the test cases failed then it will be recorded and the same will be reported to the respective development team. Sometimes, during the user acceptance testing also end users may find the error and report it to the team. All the recorded details will be reported to the respective team and they will work on the recorded errors or bugs.

ii. **Defect Mapping:**

After the error has been detected and reported to the development team, the development team will work on those errors and fix them as per the requirement. Once the development team has done its job, the tester team will again map the test cases or test scripts to that developed module or code to run the entire tests to ensure the correct output.

iii. **Re-Testing:**

From the name itself, we can easily understand that Re-Testing is the process of testing the modules or entire product again to ensure the smooth release of the module or product. In some cases, the new module or functionality will be developed after the product release. In this case, all the modules will be re-tested for a smooth release. So that it cannot cause any other defects after the release of the product or application.

iv. **Regression Testing:**

Regression Testing is software testing that ensures that the newly made changes to the code or newly developed modules or functions should not affect the normal processing of the application or product.

v. **System Integration Testing:**

System Integration Testing is a type of testing technique that will be used to check the entire component or modules of the system in a single run. It ensures that the whole system will be checked in a single test environment instead of checking each module or function separately.

Test Execution Process

i. Creation of Test Cases:

The first phase is to create suitable test cases for each module or function. Here, the tester with good domain knowledge must be required to create suitable test cases. It is always preferable to create simple test cases and the creation of test cases should not be delayed else it will cause excess time to release the product. The created test cases should not be repeated again. It should cover all the possible scenarios raised in the application.

ii. Test Cases Execution:

After test cases have been created, execution of test cases will take place. Here, the Quality Analyst team will either do automated or manual testing depending upon the test case scenario. It is always preferable to do both automated as well as manual testing to have 100% assurance of correctness. The selection of testing tools is also important to execute the test cases.

iii. Validating Test Results:

After executing the test cases, note down the results of each test case in a separate file or report. Check whether the executed test cases achieved the expected result and record the time required to complete each test case i.e., measure the performance of each test case. If any of the test cases is failed or not satisfied the condition then report it to the development team for validating the code.

Ways to Perform Test Execution

Testers can choose from the below list of preferred methods to carry out test execution:

i. Run test cases:

It is a simple and easiest approach to run test cases on the local machine and it can be coupled with other artifacts like test plans, test suites, test environments, etc.

ii. Run test suites:

A test suite is a collection of manual and automated test cases and the test cases can be executed sequentially or in parallel. Sequential execution is useful in cases where the result of the last test case depends on the success of the current test case.

iii. Run test case execution and test suite execution records:

Recording test case execution and test suite execution is a key activity in the test process and helps to reduce errors, making the testing process more efficient.

iv. Generate test results without execution:

Generating test results from non-executed test cases can be helpful in achieving comprehensive test coverage.

v. Modify execution variables:

Execution variables can be modified in the test scripts for particular test runs.

vi. Run automated and manual tests:

Test execution can be done manually or can be automated.

vii. Schedule test artifacts:

Test artifacts include video, screenshots, data reports, etc. These are very helpful as they document the results of the past test execution and provide information about what needs to be done in future test execution.

viii Defect tracking:

Without defect tracking test execution is not possible, as during testing one should be able to track the defects and identify what when wrong and where.

Test Execution Priorities

Test Execution Priorities are nothing but prioritizing the test cases depending upon several factors. It means that it executes the test cases with high efficient first than the other test cases. It depends upon various factors. Let us discuss some of the factors to be considered while prioritizing the test cases.

- **Complexity:**

The complexity of the test cases can be determined by including several factors such as boundary values of test cases, features or components of test cases, data entry of test cases, and how much the test cases cover the given business problem.

- **Risk Covered:**

How much risk that a certain test case may undergo to achieve the result. Risk in the form of time required to complete the test case process, space complexity whether it is executed in the given memory space, etc.,

- **Platforms Covered:**

It simply tells that in which platform or operating system the test cases have been executed i.e., test cases executed in the Windows OS, Mac OS, Mobile OS, etc.,

- **Depth:**

It covers how depth the given test cases cover each functionality or module in the application i.e., how much a given test procedure covers all the possible conditions in a single functionality or module.

- **Breadth:**

It covers how the breadth of the given test cases covers the entire functionality or modules in the application i.e., how much a given test procedure covers all the possible conditions in the entire functionality or modules in the product or application.

Test Execution States

The tester or the Quality Analyst team reports or notices the result of each test case and records it in their documentation or file. There are various results raised when executing the test cases. They are

- **Pass:** It tells that the test cases executed for the module or function are successful.
- **Fail:** It tells that the test cases executed for the module or function are not successful and resulted in different outputs.
- **Not Run:** It tells that the test cases are yet to be executed.
- **Partially Executed:** It tells that only a certain number of test cases are passed and others aren't met the given requirement.
- **Inconclusive:** It tells that the test cases are executed but it requires further analysis before the final submission.
- **In Progress:** It tells that the test cases are currently executed.
- **Unexpected Result:** It tells that all the test cases are executed successfully but provide different unexpected results.

Test Execution Cycle

A test execution cycle is an iterative approach that will be helpful in detecting errors. The test execution cycle includes various processes. These are:

i. **Requirement Analysis:**

In which, the QA team will gather all the necessary requirements needed for test execution. For example, how many testers are needed, what automation test tools are needed, what testing covers under the given budget, etc., the QA team will also plan depending upon the client or business requirement.

ii. **Test Planning:**

In this phase, the QA team will plan when to start and complete the testing. Choosing of correct automation test tool, and testers needed for executing the test plan. They further plan who should

develop the test cases for which module/function, who should execute the test cases, how many test cases needed to be executed, etc.,

iii. **Test Cases Development:**

This is the phase in which the QA team assigned a group of testers to write or generate the test cases for each module. A tester with good domain knowledge will easily write the best test cases or test scripts. Prioritizing the developed test cases is also the main factor.

iv. **Test Environment Setup:**

Test Environment Setup usually differs from project to project. In some cases, it is created by the team itself and it is also created by clients or customers. Test Environment Setup is nothing but testing the entire developed product with suitable software or hardware components or with both by executing all the tests on it. It is essential and it is sometimes carried out along with the test case development process.

v. **Test Execution:**

This stage involves test execution by the team and all the detected bugs are recorded and reported for remediation and rectification.

vi. **Test Closure:**

This is the final stage and here it records the entire details of the test execution process. It also contains the end-users testing details. It again modifies the testing process if any defects are found during the testing. Hence, it is a repetitive process.

Guidelines for Test Execution

1. Write the suitable test cases for each module of the function.
2. Assign suitable test cases to respective modules or functions.
3. Execute both manual testing as well as automated testing for successful results.
4. Choose a suitable automated tool for testing the application.
5. Choose the correct test environment setup.
6. Note down the execution status of each test case and note down the time taken by the system to complete the test cases.
7. Report all the success status and the failure status to the development team or to the respective team regularly.
8. Track the test status again for the already failed test cases and report it to the team.
9. Highly Skilled Testers are required to perform the testing with less or zero failures/defects.
10. Continuous testing is required until success test report is achieved.

Error/Defect Detecting and Reporting

Defect:

A defect in a software product is also known as a bug, error or fault which makes the software

produce an unexpected result as per the software requirements. For example; incorrect data, system hangs, unexpected errors, missing or incorrect requirements.

Defect Report:

A defect report is a document that has concise details about what defects are identified, what action steps make the defects show up, and what are the expected results instead of the application showing error (defect) while taking particular step by step actions.

Defect reports are usually created by the Quality Assurance team and also by the end-users (customers). Often customers detect more defects and report them to the support team of the software development since the majority of the customers curiously tries out every feature in the application. Now, you know what actually defect and defect reports are.

The reason behind why defect reports are created is to help developers to find out the defects easily and fix them up. A defect report is usually assigned by QA to a developer who then reads the report and reproduces the defects on the software product by following the action steps mentioned in the report. After that, the developer fixes the defects in order to get the desired outcome specified in the report.

That is why the defect reports are important and created carefully. Defect reports should be short, organized, straight to the point and covers all the information that the developer needs to detect the actual defects in the report by doing what and how the one written the defect report detected the defects.

It is usual for QA teams to get defect reports from the clients that are either too short to reproduce and rectify or too long to understand what actually went wrong.

For example,

Defect Description: The application doesn't work as expected. Now, how in the world does a developer or QA know what went wrong which doesn't meet the client expectation?

In such a case, the developer report to the QA that he couldn't find any problem or he may have fixed any other error but not the actual one client detected. So that's why it's really important to create a concise defect report to get bugs fixed.

All right. You have a pretty good idea about what, whys and how's of a defect report. So it's time for what is inside the report.

A typical defect report contains the information in an xls Sheet as follows.

i. **Defect ID :**

Nothing but a serial number of defects in the report.

ii. **Defect Description:**

A short and clear description of the defect detected.

iii. **Action Steps :**

What the client or QA did in an application that results in the defect. Step by step actions they took.

iv. Expected Result :

What results are expected as per the requirements when performing the action steps mentioned.

v. Actual Result:

What results are actually showing up when performing the action steps?

vi. Severity:

Trivial (A small bug that doesn't affect the software product usage).

- **Low:**

A small bug that needs to be fixed and again it's not going to affect the performance of the software.

- **Medium:**

This bug does affect the performance. Such as being an obstacle to do a certain action. Yet there is another way to do the same thing.

- **High:**

It highly impacts the software though there is a way around to successfully do what the bug cease to do.

- **Critical:**

These bugs heavily impacts the performance of the application. Like crashing the system, freezes the system or requires the system to restart for working properly.

vii. Attachments :

A sequence of screenshots of performing the step by step actions and getting the unexpected result. One can also attach a short screen recording of performing the steps and encountering defects. Short videos help developers and/or QA to understand the bugs easily and quickly.

viii. Additional information:

The platform you used, operating system and version. And other information which describes the defects in detail for assisting the developer understand the problem and fixing the code for getting desired results.

Bugs

A software bug is an error, flaw or fault in the design, development, or operation of computer software that causes it to produce an incorrect or unexpected result, or to behave in unintended ways.

The Bug is the informal name of defects, which means that software or application is not working as per the requirement. In software testing, a software bug can also be issue, error, fault, or failure. The bug occurred when developers made any mistake or error while developing the product.

While testing the application or executing the test cases, the test engineer may not get the expected result as per the requirement. And the bug had various names in different companies such as error, issues, problem, fault, and mistake, etc.

Basic terminology of defect

Let see the different terminology of defect:

a. Defect

b. Bug

c. Error

d. Issue

e. Mistakev

f. Failurev

Reason for Occurring Bugs

In software testing, the bug can occur for the following reasons:

a. Wrong coding

b. Missing coding

c. Extra coding

i. Wrong Coding

Wrong coding means improper implementation.

For example: Suppose if we take the Gmail application where we click on the "Inbox" link, and it navigates to the "Draft" page, this is happening because of the wrong coding which is done by the developer, that's why it is a bug.

ii. Missing Coding

Here, missing coding means that the developer may not have developed the code only for that particular feature.

For example: if we take the above example and open the inbox link, we see that it is not there only, which means the feature is not developed only.

iii. Extra Coding

Here, extra coding means that the developers develop the extra features, which are not required according to the client's requirements.

For example: Suppose we have one application form wherein the Name field**,** the First name**,** and Last name textbox are needed to develop according to the client's requirement.

But, the developers also develop the **"Middle name"** textbox, which is not needed according to the client's requirements as we can see in the below Figure 5.5:

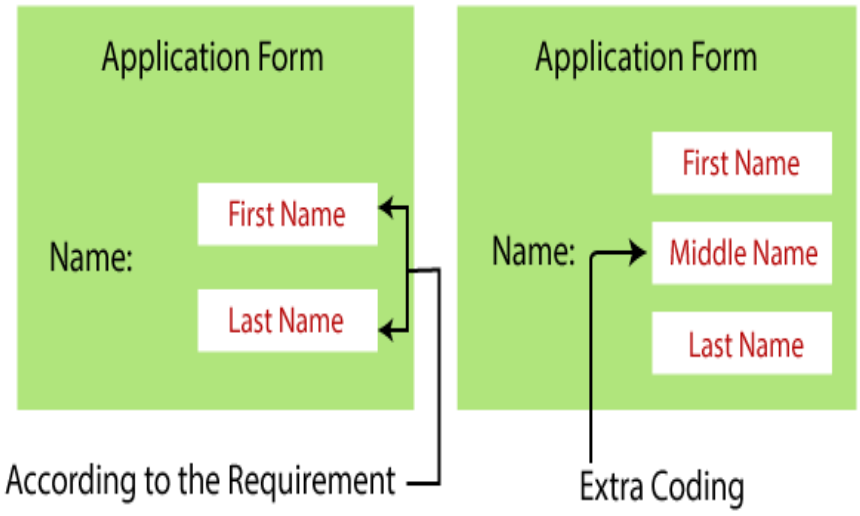


Figure 5.5: Represented that the Different Application Form.

If we develop an extra feature that is not needed in the requirement, it leads to unnecessary extra effort. And it might also happen that adding up the extra feature affects the other elements too.

Bug tracking tool

We have various types of bug tracking tools available in software testing that helps us to track the bug, which is related to the software or the application. Some of the most commonly used bug tracking tools are as follows:

1. Jira
2. Bugzilla
3. Redmine
4. Mantis
5. Backlog

i. Jira

Jira is one of the most important bug tracking tools. Jira is an open-source tool that is used for bug tracking, project management, and issue tracking in manual testing.

Jira includes different features like reporting, recording, and workflow. In Jira, we can track all kinds of bugs and issues, which are related to the software and generated by the test engineer.

To get the complete details about Jira tool, refer to the below link:

ii. Bugzilla

Bugzilla is another important bug tracking tool, which is most widely used by many organizations to track the bugs. Bugzilla is an open-source tool, which is used to help the customer, and the client to maintain the track of the bugs.

It is also used as a test management tool because, in this, we can easily link other test case management tools such as ALM, quality Centre, etc.

Bugzilla supports various operating systems such as Windows, Linux, and Mac. Bugzilla has some features which help us to report the bug easily:

- A bug can be list in multiple formats
- Email notification controlled by user preferences.
- Advanced searching capabilities
- Excellent security
- Time tracking

iii. Redmine

It is an open-source tool which is used to track the issues and web-based project management tool. Redmine tool is written in **Ruby** programing language and also compatible with multiple databases like MySQL, Microsoft SQL, and SQLite. While using the Redmine tool, users can also manage the various project and related subprojects. Some of the common characteristics of Redmine tools are as follows:

- Flexible role-based access control
- Time tracking functionality
- A flexible issue tracking system
- Feeds and email notification
- Multiple languages support (Albanian, Arabic, Dutch, English, Danish and so on)

iv. MantisBT

MantisBT stands for Mantis Bug Tracker**.** It is a web-based bug tracking system, and it is also an open-source tool. MantisBT is used to follow the software defects. It is executed in the PHP programing language. Some of the common features of MantisBT are as follows:

- Full-text search
- Audit trails of changes made to issues
- Revision control system integration
- Revision control of text fields and notes

- Notifications
- Plug-ins
- Graphing of relationships between issues

Backlog

The backlog is widely used to manage the IT projects and track the bugs. It is mainly built for the development team for reporting the bugs with the complete details of the issues, comments. Updates and change of status. It is a project management software.

Features of backlog tool are as follows:

- Gantt and burn down charts
- It supports Git and SVN repositories
- IP access control
- Support Native iOS and Android apps

Debugging

To launch an application into the market, it is very necessary to cross-check it multiple times so as to deliver an error-free product. When we talk about delivering a bug-free product, then our main concern is all about customer satisfaction because if you are application is not up to the mark, then eventually it will demolish the company's reputation in the market. In this article, we are going to see what makes debugging stand out of the queue and how it is different from software testing. We will be discussing the following topics:

In the development process of any software, the software program is religiously tested, troubleshot, and maintained for the sake of delivering bug-free products. There is nothing that is error-free in the first go. So, it's an obvious thing to which everyone will relate that as when the software is created, it contains a lot of errors; the reason being nobody is perfect and getting error in the code is not an issue, but avoiding it or not preventing it, is an issue!

All those errors and bugs are discarded regularly, so we can conclude that debugging is nothing but a process of eradicating or fixing the errors contained in a software program. Debugging works stepwise, starting from identifying the errors, analyzing followed by removing the errors. Whenever a software fails to deliver the result, we need the software tester to test the application and solve it.

Since the errors are resolved at each step of debugging in the software testing, so we can conclude that it is a tiresome and complex task regardless of how efficient the result was.

Need Debugging

Debugging gets started when we start writing the code for the software program. It progressively starts continuing in the consecutive stages to deliver a software product because the code gets

merged with several other programming units to form a software product. Following are the benefits of Debugging:

- Debugging can immediately report an error condition whenever it occurs. It prevents hampering the result by detecting the bugs in the earlier stage, making software development stress-free and smooth.
- It offers relevant information related to the data structures that further helps in easier interpretation.
- Debugging assist the developer in reducing impractical and disrupting information.
- With debugging, the developer can easily avoid complex one-use testing code to save time and energy in software development.

Steps involved in Debugging

Following are the different steps that are involved in debugging, as shown in below Figure 5.6:

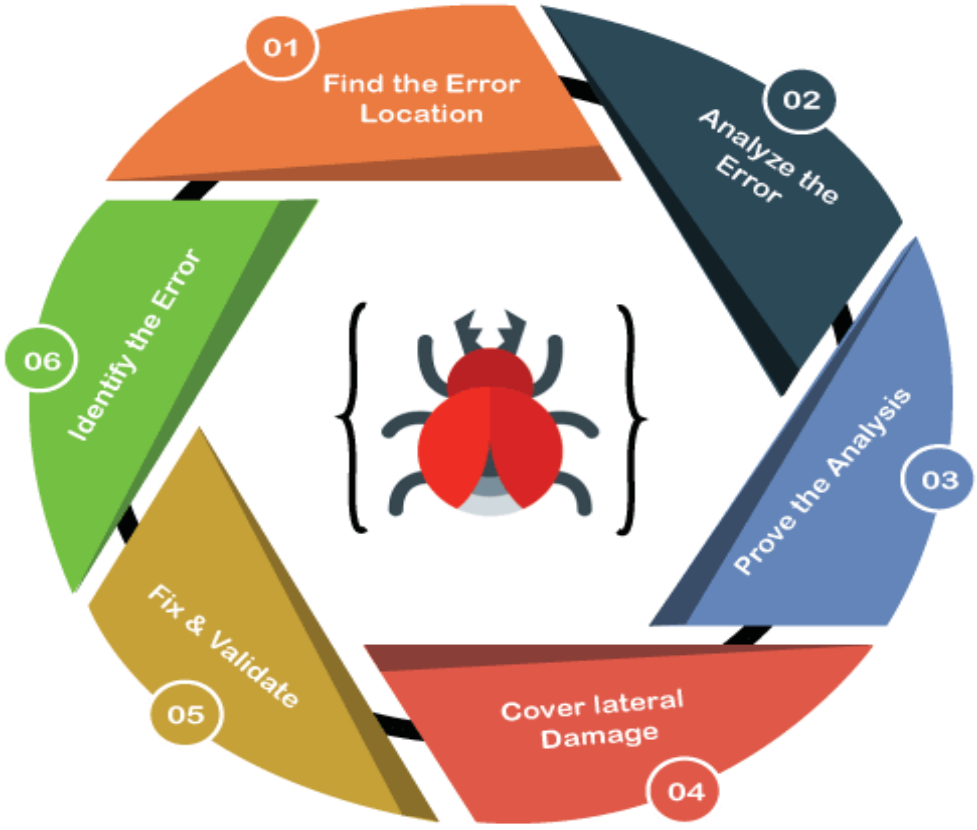


Figure 5.6: Represented that the Debugging Process.

i. **Identify the Error:**

Identifying an error in a wrong may result in the wastage of time. It is very obvious that the production errors reported by users are hard to interpret, and sometimes the information we receive is misleading. Thus, it is mandatory to identify the actual error.

ii. **Find the Error Location:**

Once the error is correctly discovered, you will be required to thoroughly review the code repeatedly to locate the position of the error. In general, this step focuses on finding the error rather than perceiving it.

iii. **Analyze the Error:**

The third step comprises error analysis, a bottom-up approach that starts from the location of the error followed by analyzing the code.

This step makes it easier to comprehend the errors. Mainly error analysis has two significant goals, i.e., evaluation of errors all over again to find existing bugs and postulating the uncertainty of incoming collateral damage in a fix.

iv. **Prove the Analysis:**

After analyzing the primary bugs, it is necessary to look for some extra errors that may show up on the application. By incorporating the test framework, the fourth step is used to write automated tests for such areas.

v. **Cover Lateral Damage:**

The fifth phase is about accumulating all of the unit tests for the code that requires modification. As when you run these unit tests, they must pass.

vi. **Fix & Validate:**

The last stage is the fix and validation that emphasizes fixing the bugs followed by running all the test scripts to check whether they pass.

Debugging Strategies

- For a better understanding of a system, it is necessary to study the system in depth. It makes it easier for the debugger to fabricate distinct illustrations of such systems that are needed to be debugged.
- The backward analysis analyzes the program from the backward location where the failure message has occurred to determine the defect region. It is necessary to learn the area of defects to understand the reason for defects.
- In the forward analysis, the program tracks the problem in the forward direction by utilizing the breakpoints or print statements incurred at different points in the program. It emphasizes those regions where the wrong outputs are obtained.
- To check and fix similar kinds of problems, it is recommended to utilize past experiences. The success rate of this approach is directly proportional to the proficiency of the debugger.

Debugging Tools

The debugging tool can be understood as a computer program that is used to test and debug several other programs.

Presently, there are many public domain software such as **gdb** and **dbx** in the market, which can be utilized for debugging. These software offers console-based command-line interfaces. Some of the automated debugging tools include code-based tracers, profilers, interpreters, etc.

Figure 5.7: Represented that the Debugging Tools.

Here is a list of some of the widely used debuggers:

- Radare2
- WinDbg
- Valgrind

Radare2

Radare2 is known for its reverse engineering framework as well as binary analysis. It is made up of a small set of utilities, either utilized altogether or independently from the command line. It is also known as r2. It is constructed around disassembler for computer software for generating assembly language source code from machine-executable code. It can support a wide range of executable formats for distinct architectures of processors and operating systems.

WinDbg

WinDbg is a multipurpose debugging tool designed for Microsoft Windows operating system. This tool can be used to debug the memory dumps created just after the Blue Screen of Death that further arises when a bug check is issued. Besides, it is also helpful in debugging the user-mode crash dumps, which is why it is called post-mortem debugging.

Valgrind

The Valgrind exist as a tool suite that offers several debugging and profiling tools to facilitate users in making faster and accurate program. Memcheck is one of its most popular tools, which can successfully detect memory-related errors caused in C and C++ programs as it may crash the program and result in unpredictable behavior.

Debugging Approaches

Need for debugging:

Once errors are known during a program code, it's necessary to initial establish the precise program statements liable for the errors and so to repair them.

Challenges in Debugging:

There are lot of problems at the same time as acting the debugging. These are the following:

- Debugging is finished through the individual that evolved the software program and it's miles difficult for that person to acknowledge that an error was made.
- Debugging is typically performed under a tremendous amount of pressure to fix the supported error as quick as possible.
- It can be difficult to accurately reproduce input conditions.
- Compared to the alternative software program improvement activities, relatively little research, literature and formal preparation exist at the procedure of debugging.

Debugging Approaches:

The following are a number of approaches popularly adopted by programmers for debugging.

- **Brute Force Method:**

This is the foremost common technique of debugging however that is the least economical method. During this approach, the program is loaded with print statements to print the intermediate values with the hope that a number of the written values can facilitate to spot the statement in error. This approach becomes a lot of systematic with the utilisation of a symbolic program (also known as a source code debugger), as a result of values of various variables will be simply checked and breakpoints and watch-points can be easily set to check the values of variables effortlessly.

- **Backtracking:**

This is additionally a reasonably common approach. During this approach, starting from the statement at which an error symptom has been discovered, the source code is derived backward till the error is discovered. Sadly, because the variety of supply lines to be derived back will increase, the quantity of potential backward methods will increase and should become unimaginably large so limiting the utilization of this approach.

- **Cause Elimination Method:**

In this approach, a listing of causes that may presumably have contributed to the error symptom is developed and tests are conducted to eliminate every error. A connected technique of identification of the error from the error symptom is that the package fault tree analysis.

- **Program Slicing:**

This technique is analogous to backtracking. Here the search house is reduced by process slices. A slice of a program for a specific variable at a particular statement is that the set of supply lines preceding this statement which will influence the worth of that variable.

Debugging Guidelines:

Debugging is commonly administrated by programmers supported their ingenuity. The subsequent are some general tips for effective debugging:

- Many times debugging needs an intensive understanding of the program style. Making an attempt to rectify supported a partial understanding of the system style and implementation might need an excessive quantity of effort to be placed into debugging even straightforward issues.
- Debugging might generally even need a full plan of the system. In such cases, a typical mistake that novice programmers usually create is trying to not fix the error however its symptoms.
- One should be watched out for the likelihood that a slip correction might introduce new errors. So, when each spherical of error-fixing, regression testing should be administrated.

Defect Report

A defect in a software product is also known as a bug, error or fault which makes the software produce an unexpected result as per the software requirements. For example; incorrect data, system hangs, unexpected errors, missing or incorrect requirements. A defect report is a document that has concise details about what defects are identified, what action steps make the defects show up, and what are the expected results instead of the application showing error (defect) while taking particular step by step actions.

Defect reports are usually created by the Quality Assurance team and also by the end-users (customers). Often customers detect more defects and report them to the support team of the software development since the majority of the customers curiously tries out every feature in the application. Now, you know what actually defect and defect reports are:

The reason behind why defect reports are created is to help developers to find out the defects easily and fix them up. A defect report is usually assigned by QA to a developer who then reads the report and reproduces the defects on the software product by following the action steps mentioned in the report. After that, the developer fixes the defects in order to get the desired outcome specified in the report.

That is why the defect reports are important and created carefully. Defect reports should be short, organized, straight to the point and covers all the information that the developer needs to detect the actual defects in the report by doing what and how the one written the defect report detected the defects. It is usual for QA teams to get defect reports from the clients that are either too short to reproduce and rectify or too long to understand what actually went wrong.

For example,

Defect Description: The application doesn't work as expected.

Now, how in the world does a developer or QA know what went wrong which doesn't meet the client expectation?

In such a case, the developer report to the QA that he couldn't find any problem or he may have fixed any other error but not the actual one client detected. So that's why it's really important to create a concise defect report to get bugs fixed.

All right. You have a pretty good idea about what, whys and how's of a defect report. So it's time for what is inside the report.

A typical defect report contains the information in an xls Sheet as follows.

i. **Defect ID:** Nothing but a serial number of defects in the report.

ii. **Defect Description:** A short and clear description of the defect detected.

iii. **Action Steps:** What the client or QA did in an application that results in the defect. Step by step actions they took.

iv. **Expected Result:** What results are expected as per the requirements when performing the action steps mentioned.

v. **Actual Result:** What results are actually showing up when performing the action steps.

vi. **Severity:** Trivial (A small bug that doesn't affect the software product usage).

- **Low:** A small bug that needs to be fixed and again it's not going to affect the performance of the software.
- **Medium:** This bug does affect the performance. Such as being an obstacle to do a certain action. Yet there is another way to do the same thing.
- **High:** It highly impacts the software though there is a way around to successfully do what the bug cease to do.
- **Critical:** These bugs heavily impacts the performance of the application. Like crashing the system, freezes the system or requires the system to restart for working properly.

vii. **Attachments:**

A sequence of screenshots of performing the step by step actions and getting the unexpected result. One can also attach a short screen recording of performing the steps and encountering defects. Short videos help developers and/or QA to understand the bugs easily and quickly.

viii. **Additional information:**

The platform you used, operating system and version. And other information which describes the defects in detail for assisting the developer understand the problem and fixing the code for getting desired results.

Severity in Testing

One can define Severity as the extent to which any given defect can affect/ impact a particular software. Severity is basically a parameter that denotes the impact of any defect and its implication on a software's functionality. In other words, Severity defines the overall impact that any defect can have on a system. For instance, consider if a web page or an application crashes after clicking on a remote link. In such a case, a user would rarely click on the remote link. Yet, the overall impact of an app crashing is very severe. Hence, the severity gets high, and yet the priority gets low.

Priority in Testing

One can define Priority as a parameter for deciding the order in which one can fix the defect. In this, the defect with a higher priority first needs to get fixed. Priority basically defines the order in

which one would resolve any given defect. The priority status defines if we should fix something or wait. The tester sets this priority status to the developer along with mentioning a time frame that can fix that defect. If they mention a higher priority, then the developer needs to fix it at the very earliest. Basically, the priority status comes into play according to the customer's requirements.

For instance, let's consider a case where one misspells the company name on a website's home page. Thus, in this case, the Priority gets high while the Severity gets low for fixing it as display in Table 5.2.

Table 5.2: Represented that the Difference between Severity and Priority in Testing

Parameters	Severity in Testing	Priority in Testing
Definition	Severity is a term that denotes how severely a defect can affect the functionality of the software.	Priority is a term that defines how fast we need to fix a defect.
Parameter	Severity is basically a parameter that denotes the total impact of a given defect on any software.	Priority is basically a parameter that decides the order in which we should fix the defects.
Relation	Severity relates to the standards of quality.	Priority relates to the scheduling of defects to resolve them in software.
Value	The value of severity is objective.	The value of priority is subjective.
Change of Value	The value of Severity changes continually from time to time.	The value of Priority changes from time to time.
Who Decides the Defect	The testing engineer basically decides a defect's severity level.	The product manager basically decides a defect's priority level.
Types	There are 5 types of Severities: Cosmetic, Minor, Moderate, Major, and Critical.	There are 3 types of Priorities: High, Medium, and Low.

Test Closure

Software testing is the process of evaluating and analyzing whether the software product is working as expected and doing what it is supposed to do. There are many benefits of performing the testing activity like improved software quality, reduced bugs, reduced development costs, etc. The end result of the testing process is Test Closure. The article focuses on discussing the test closure.

Test Closure is a document that provides a summary of all the tests covered during the software development lifecycle. It includes various activities like test completion reporting, a summary of test results as well as the test completion matrix. It gives us an outline of the tests conducted during the software testing, and details of the errors and bugs found and resolved during the testing phase. In other words, it can be said that a Test Closure is a memo that the testing team prepares prior to officially finishing the testing process.

a. The Test Cycle Closure phase is the end goal of execution of the testing phase that includes the summary of all the tests conducted during the software development life cycle.

b. It is a formal document that includes the objectives met, total time taken, total costs, test coverage and bugs/error found and resolved.

c. Members of the testing team meet to discuss and analyze testing artifacts to identify plans that have been implemented in future cases, learning from the current life cycle.

d. The main purpose of the test closure phase is to extract process bottlenecks of software development life cycles in future.

Need for Test Closure

a. Helps to create consolidated test results report:

In the test closure process, all the results of the tests that were performed are prepared. In this way, the QA team is able to detect the errors and bugs in the software system. All these errors/bugs are documented and can be referred to for future releases.

b. The official announcement of the end of the testing phase:

The test closure process proves to be a good medium for other members to know about the end of the testing phase.

c. Provide detailed analysis of the error:

Test closure provides a complete report of all the errors discovered and resolved so the source of origin can be located in case of error and resolution can be provided.

d. Showcase the metrics to the customer:

Presenting test metrics to the customer representatives, will help them understand more about the software system, the details of the features, the number of errors discovered, and how they were fixed.

e. Assessment of Risk Factor:

The most important part of the test closure process is the assessment of the risk factor with respect to the software system as a whole. This also helps the client to know the strength as well as weaknesses of the software program.

Stages of Test Closure

The test closure process is executed in the following six stages, which helps the team to ensure that the software system being deployed is error free and well effective for the end users. The stages involved in the test closure process are as follows:

- **Analyze and Check Planned Deliverables:**

The testing team checks and analyzes the required documents including the test data, test plan, test strategy, traceability matrix, and test completion report that have to be delivered to the stakeholder of the project.

- **Closure of Incident Reports:**

During the testing process, the testing team observes certain variations and deviations that are mentioned in the incident report. This incident report is prepared in the earlier stages of testing. In this stage, the team checks and ensures that all the incidents mentioned in the incident report are closed.

- **Handover to Maintenance Team:**

After the incident reports are closed, the testing team hands over the testwares to the maintenance team.

- **Finalizing Testwares:**

In this stage, the testwares like test cases, test scripts, etc are finalized for future release and use by clients.

- **Document System Acceptance:**

In this stage, the client accepts the developed software with the entire details and agrees to use it. This also includes the validation and verification of the software system as defined in the validation section, before the actual implementation of the system software.

- **Analyze Best Practices:**

This is the last stage of test closure. In this stage, the required data and information is collected to enhance the maturity of testwares and lessons to be remembered for future releases. This aids the team to determine modifications that are required for similar future releases.

Test Closure Activities

When the testing process of a particular software system gets completed, certain activities are performed, which include the collection of the results of the test cases that were executed and, either handing over the results to the concerned person or archiving the results. These activities are performed collectively and are termed Test Closure Activities.

These activities are performed after the testing phase is complete and after the software release. The main purpose behind these activities is to ensure the quality of the software. With the help of these activities, the actual figures and facts can also be recognized during the project life cycle.

There are four types of Test Closure Activities:

- **Validating completion of test cases:**

During this activity, the manager or the lead of the testing team validates that all test work has been completed. The team manager is responsible to cross-check all the documents like test cases, test plans, and test strategies to make sure that nothing is left out and all the known bugs or errors are either resolved, postponed, or admitted as a permanent limitations.

- **Handing over test artifacts:**

After the test work is validated by the manager of the testing team, they are handed over to the concerned people, who require them in the future.

- **Defining Learning Experience:**

The most important aspect of the test closure activity is the learning experience that the team gains from their participation in the team meetings. In these meetings, proper planning is done to ensure that the best practices that were used are followed for future releases, while the poor practices leading to unfavorable outcomes are eliminated and not repeated in future

- **Archiving Test Work:**

The final task of the testing team is to archive all the significant test work, documents, and reports, logs of the tests, test artifacts, test plans, and work of configuration management. The planning archive must store both the test plan as well as the project plan, with a significant linkage to the version and software system of the product.

Test Closure Activities

These test closure activities are performed when the testing phase is completed. There are certain conditions that mark the completion of the testing process and thus the team needs to perform these activities.

The conditions are as follows:

- **After Identification of errors/bugs:**

When the testing team acknowledges the errors and bugs in the software system, along with their source of origin. Then test closure activities are performed.

- **Achievement of target:**

When the testing team is able to achieve the particular goal or target, as desired by the client, this marks the completion of testing, and thus, test closure activities need to be performed.

- **When modifications are required:**

Test Closure activities are used when some modifications are required to be done.

- **Lessons learned are completed:**

The completion and documentation of learning experiences mark the completion of the testing process and thus the team needs to perform test closure activities

- **Maintenance:**

Test Closure activities are done when some maintenance work is required to be done

- **Project Cancellation:**

When some project gets canceled due to specific reasons, test closure activities are performed.

The Entry Criteria for test closure:

The entry criteria in testing give us all the necessary conditions that need to be satisfied before the actual testing can begin. It involves:

- Test cases are completed
- We have obtained the results of the test cases.
- We have obtained the logs of the defects

- Reports of errors/bugs found are ready

The Exit Criteria for test closure:

The entry criteria in testing give us all the necessary documents or items that need to be completed before the actual testing can end. It involves:

- The test Closure report has been delivered.
- The report has been approved as well as signed off officially by the client.

Test Closure Report

A test closure report is a formal document that contains a summary of all test cases of a testing project and their final test results.

The test report is an assessment of how well the testing is performed. Based on the test report the stakeholders can examine the quality of the tested product and make a decision on its release. After completion of testing, various matrices are collected to prepare the test reports.

The criteria for preparing the test closure report include that the test results should be saved and can be produced along with the test cycle closure documents to support the completion of test execution. Documents must contain screenshots, query results, recordings as well as log files. The test closure report includes the following:

1. Test Summary Report.
2. Identifier.
3. Test Summary.
4. Variances.
5. Comprehensive Assessment.
6. Summary of Results.
7. Evaluation.
8. Summary of Activities.
9. Approval.

Benefits of Test Closure Activities

a) **Validates the delivery of deliverables:**
The test closure activities help to check whether the planned deliverables are properly delivered to the client or end users or not.

b) **Usability:**
The test closure activities ensures that the software product meets the requirement of the end-users

c) **Functional Correctness:**
The test closure activities ensure that all kinds of bugs and errors are documented in a report and resolved. It also helps to take care that these do not get repeated in the future.

d) **Positive Impact for future:**
The test closure activities examine the total figures achieved in the whole process of development and testing, which creates a positive impact for future references.

e) **Delivery of important artifacts:**
The test closure activities permit the manager of the testing team to pass important test cases and artifacts to the relevant person.

f) **Evaluation of testing:**
The test closure activities help to evaluate the testing process. It also helps in the analysis of lessons learned for testing as well as development

---

CHAPTER 6

USE OF TEST METRICS IN SOFTWARE TESTING

Dr. Santosh S Chowhan, Assistant Professor
Department of Data Science & Analytics, School of Sciences, Jain (Deemed-to-be University), Bangalore-27, India
Email Id- santosh.sc@jainuniversity.ac.in

Software Testing Metrics are the quantitative measures used to estimate the progress, quality, productivity and health of the software testing process. The goal of software testing metrics is to improve the efficiency and effectiveness in the software testing process and to help make better decisions for further testing process by providing reliable data about the testing process. A Metric which id display in Figure 6.1, defines in quantitative terms the degree to which a system, system component, or process possesses a given attribute, which is given below figure. The ideal example to understand metrics would be a weekly mileage of a car compared to its ideal mileage recommended by the manufacturer.

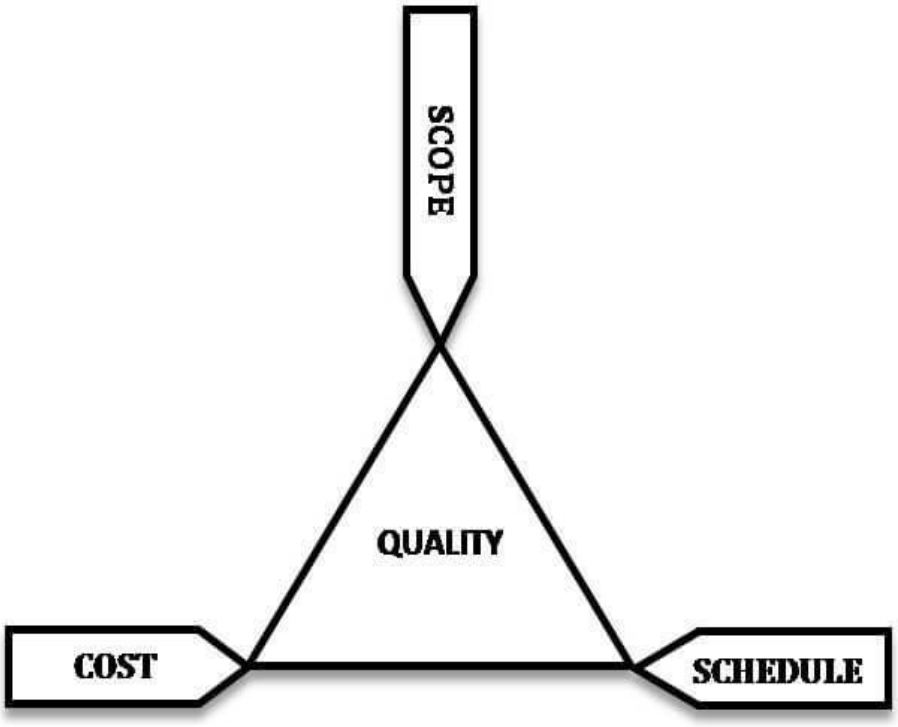


Figure 6.1: Represented that the Quality of Test Metrics.

Software testing metrics or software test measurement is the quantitative indication of extent, capacity, dimension, amount or size of some attribute of a process or product.

Importance of Test Matrices

We cannot improve what we cannot measure" and Test Metrics helps us to do exactly the same.

A. Take decision for next phase of activities

B. Evidence of the claim or prediction
C. Understand the type of improvement required
D. Take decision or process or technology change

Types of Test Metrics

Types of Matrix are show through the Figure 6.2 are following:

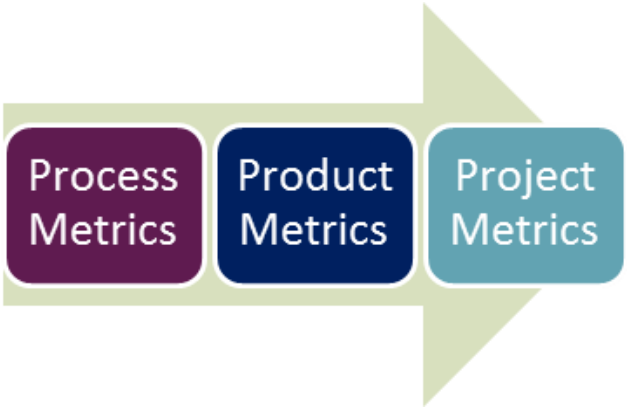


Figure 6.2: Displayed that the Types of Matrix.

A. **Process Metrics:** It can be used to improve the process efficiency of the SDLC (Software Development Life Cycle)
B. **Product Metrics:** It deals with the quality of the software product
C. **Project Metrics:** It can be used to measure the efficiency of a project team or any testing tools being used by the team members

Identification of correct testing metrics is very important. Few things need to be considered before identifying the test metrics

A. Fix the target audience for the metric preparation
B. Define the goal for metrics
C. Introduce all the relevant metrics based on project needs
D. Analyze the cost benefits aspect of each metrics and the project lifestyle phase in which it results in the maximum output

Test Metrics Life Cycle

In the Table 6.1, there is display the life cycle in different stages of Test Metrics.

Table 6.1: Represented that the Life Cycle of Test Metrics.

Different stages of Metrics life cycle	**Steps during each stage**
• Analysis	• Identification of the Metrics

	• Define the identified QA Metrics
• Communicate	• Explain the need for metric to stakeholder and testing team • Educate the testing team about the data points to need to be captured for processing the metric
• Evaluation	• Capture and verify the data • Calculating the metrics value using the data captured
• Report	• Develop the report with an effective conclusion • Distribute the report to the stakeholder and respective representative • Take feedback from stakeholder

Working of Manual Test Metrics

Manual testing is carried out in a step-by-step manner by quality assurance experts. Test automation frameworks, tools, and software are used to execute tests in automated testing. There are advantages and disadvantages to both human and automated testing. Manual testing is a time-consuming technique, but it allows testers to deal with more complicated circumstances. There are two sorts of manual test metrics:

i. Base Metrics:

Analysts collect data throughout the development and execution of test cases to provide base metrics. By generating a project status report, these metrics are sent to test leads and project managers. It is quantified using calculated metrics.

A. The total number of test cases

B. The total number of test cases completed.

ii. Calculated Metrics:

Data from base metrics are used to create calculated metrics. The test lead collects this information and transforms it into more useful information for tracking project progress at the module, tester, and other levels. It's an important aspect of the SDLC since it allows developers to make critical software changes.

Other Important Metrics

The following are some of the other important software metrics:

A. **Defect metrics:** Defect metrics help engineers understand the many aspects of software quality, such as functionality, performance, installation stability, usability, compatibility, and so on.

B. **Schedule Adherence:** Schedule Adherence's major purpose is to determine the time difference between a schedule's expected and actual execution times.

C. **Defect Severity:** The severity of the problem allows the developer to see how the defect will affect the software's quality**.**

D. **Test case efficiency:** Test case efficiency is a measure of how effective test cases are at detecting problems.

E. **Defects finding rate:** It is used to determine the pattern of flaws over a period of time.

F. **Defect Fixing Time:** The amount of time it takes to remedy a problem is known as defect fixing time.

G. **Test Coverage:** It specifies the number of test cases assigned to the program. This metric ensures that the testing is completed completely. It also aids in the verification of code flow and the testing of functionality.

H. **Defect cause:** It's utilized to figure out what's causing the problem.

Test Metrics Life Cycle

The below diagram illustrates the different stages in the test metrics life cycle, mention in below Figure 6.3.

The various stages of the test metrics lifecycle are:

i. **Analysis:**

- The metrics must be recognized.
- Define the QA metrics that have been identified.

ii. **Communicate:**

- Stakeholders and the testing team should be informed about the requirement for metrics.
- Educate the testing team on the data points that must be collected in order to process the metrics.

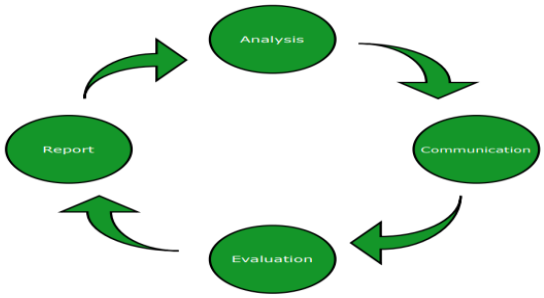


Figure 6.3: Represented that the Test Metrics Life Cycle

iii. **Evaluation:**

- Data should be captured and verified.
- Using the data collected to calculate the value of the metrics

iv. **Report:**

- Create a strong conclusion for the paper.
- Distribute the report to the appropriate stakeholder and representatives.
- Gather input from stakeholder representatives.

Formula for Test Metrics

- To get the percentage execution status of the test cases, the following formula can be used:

Percentage test cases executed = (No of test cases executed / Total no of test cases written) x 100

Similarly, it is possible to calculate for other parameters also such as test cases that were not executed, test cases that were passed, test cases that were failed, and test cases that were blocked, and so on. Below are some of the formulas:

1. Test Case Effectiveness:

Test Case Effectiveness = (Number of defects detected / Number of test cases run) x 100

2. Passed Test Cases Percentage: Test Cases that Passed Coverage is a metric that indicates the percentage of test cases that pass.

Passed Test Cases Percentage = (Total number of tests ran / Total number of tests executed) x 100

3. Failed Test Cases Percentage: This metric measures the proportion of all failed test cases.

Failed Test Cases Percentage = (Total number of failed test cases / Total number of tests executed) x 100

4. Blocked Test Cases Percentage: During the software testing process, this parameter determines the percentage of test cases that are blocked.

Blocked Test Cases Percentage = (Total number of blocked tests / Total number of tests executed) x 100

5. Fixed Defects Percentage: Using this measure, the team may determine the percentage of defects that have been fixed.

Fixed Defects Percentage = (Total number of flaws fixed / Number of defects reported) x 100

6. Rework Effort Ratio: This measure helps to determine the rework effort ratio.

Rework Effort Ratio = (Actual rework efforts spent in that phase/ Total actual efforts spent in that phase) x 100

7. Accepted Defects Percentage: This measures the percentage of defects that are accepted out of the total accepted defects.

Accepted Defects Percentage = (Defects Accepted as Valid by Dev Team / Total Defects Reported) x 100

8. Defects Deferred Percentage: This measures the percentage of the defects that are deferred for future release.

Defects Deferred Percentage = (Defects deferred for future releases / Total Defects Reported) x 100

Importance of Metrics in Software Testing

Test metrics are essential in determining the software's quality and performance. Developers may use the right software testing metrics to improve their productivity.

- Test metrics help to determine what types of enhancements are required in order to create a defect-free, high-quality software product.
- Make informed judgments about the testing phases that follow, such as project schedule and cost estimates.
- Examine the current technology or procedure to see if it need any more changes.

Quality Assurance (QA) & Quality Control (QC) & Testing

Quality Assurance

Quality Assurance (SQA) is simply a way to assure quality in the software. It is the set of activities which ensure processes, procedures as well as standards are suitable for the project and implemented correctly.

Software Quality Assurance is a process which works parallel to development of software. It focuses on improving the process of development of software so that problems can be prevented before they become a major issue. Software Quality Assurance is a kind of Umbrella activity that is applied throughout the software process.

Generally the quality of the software is verified by the third party organization like international standard organizations.

Software quality assurance focuses on:

a. Software's portability

b. Software's usability

c. Software's reusability

d. Software's correctness

e. Software's maintainability

f. Software's error control

Software Quality Assurance has:

a. A quality management approach

b. Formal technical reviews

c. Multi testing strategy

d. Effective software engineering technology

e. Measurement and reporting mechanism

Major Software Quality Assurance Activities:

i. **SQA Management Plan:**

Make a plan for how you will carry out the sqa throughout the project. Think about which set of software engineering activities are the best for project. Check level of sqa team skills.

ii. **Set the Check Points:**

SQA team should set checkpoints. Evaluate the performance of the project on the basis of collected data on different check points.

iii. **Multi testing Strategy:**

Do not depend on a single testing approach. When you have a lot of testing approaches available use them.

iv. **Measure Change Impact:**

The changes for making the correction of an error sometimes re introduces more errors keep the measure of impact of change on project. Reset the new change to change check the compatibility of this fix with whole project.

v. **Manage Good Relations:**

In the working environment managing good relations with other teams involved in the project development is mandatory. Bad relation of sqa team with programmers' team will impact directly and badly on project. Don't play politics.

Benefits of Software Quality Assurance (SQA):

1. SQA produces high quality software.
2. High quality application saves time and cost.
3. SQA is beneficial for better reliability.

4. SQA is beneficial in the condition of no maintenance for a long time.
5. High quality commercial software increase market share of company.
6. Improving the process of creating software.
7. Improves the quality of the software.

Disadvantage of SQA:

There are a number of disadvantages of quality assurance. Some of them include adding more resources, employing more workers to help maintain quality and so much more.

Quality Control

Quality control is a set of methods used by organizations to achieve quality parameters or quality goals and continually improve the organization's ability to ensure that a software product will meet quality goals.

Quality Control Process:

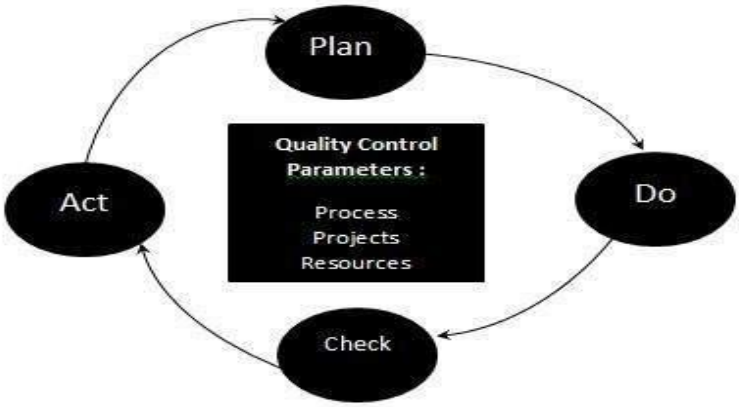


Figure 6.4: Displayed the Process of Quality Control.

The three class parameters that control software quality are display in above Figure 6.4:

- Products
- Processes
- Resources

The total quality control process consists of:

- Plan - It is the stage where the Quality control processes are planned
- Do - Use a defined parameter to develop the quality
- Check - Stage to verify if the quality of the parameters are met
- Act - Take corrective action if needed and repeat the work

Quality Control characteristics:

- Process adopted to deliver a quality product to the clients at best cost.

- Goal is to learn from other organizations so that quality would be better each time.
- To avoid making errors by proper planning and execution with correct review process.

Data Flow Testing

Data flow testing is an unfortunate term because it suggests some connection with data flow diagrams; no connection exists. Data flow testing refers to forms of structural testing that focus on the points at which variables receive values and the points at which these values are used (or referenced). We will see that data flow testing serves as a "reality check" on path testing; indeed, many of the data flow testing proponents and researchers see this approach as a form of path testing. While dataflow and slice-based testing are cumbersome at the unit level; they are well suited for object-oriented code. We will look at two mainline forms of data flow testing: one provides a set of basic definitions and a unifying structure of test coverage metrics, while the other is based on a concept called a "program slice." Both of these formalize intuitive behaviors and analyses of testers; and although they both start with a program graph, both move back in the direction of functional testing. Also, both of these methods are difficult to perform manually, and unfortunately, few commercial tools exist to make life easier for the data flow and slicing testers. On the positive side, both techniques are helpful for coding and debugging. Most programs deliver functionality in terms of data. Variables that represent data somehow receive values, and these values are used to compute values for other variables. Since the early 1960s, programmers have analyzed source code in terms of the point's statements and statement fragments at which variables receive values and points at which these values are used. Many times, their analyses were based on concordances that list statement numbers in which variable names occur. Concordances were popular features of second-generation language compilers they are still popular with COBOL programmers. Early data flow analyses often centered on a set of faults that are now known as define/reference anomalies:

- A variable that is defined but never used,
- A variable that is used before it is defined,
- A variable that is defined twice before it is used

Each of these anomalies can be recognized from the concordance of a program. Because the concordance information is compiler generated, these anomalies can be discovered by what is known as static analysis: finding faults in source code without executing it.

i. **Define/Use Testing:**

Much of the formalization of define/use testing; the definitions in this section are compatible with those, which summarizes most define/use testing theory. It presumes a program graph in which nodes are statement fragments a fragment may be an entire statement and programs that follow the structured programming precepts. The following definitions refer to a program P that has a program graph G(P) and a set of program variables V. The program graph G(P) is constructed as in Chapter 4, with statement fragments as nodes and edges that represent node sequences. G(P) has a single-entry node and a single-exit node. We also disallow edges from a node to itself. Paths, subpaths, and cycles are as they were in Chapter 4. The set of all paths in P is PATHS(P).

- Definition 1:

Node n ∈ G(P) is a defining node of the variable v ∈ V, written as DEF(v, n), if and only if the value of variable v is defined as the statement fragment corresponding to node n. Input statements, assignment statements, loop control statements, and procedure calls are all examples of statements that are defining nodes. When the code corresponding to such statements executes, the contents of the memory location(s) associated with the variables are changed.

- Definition 2:

Node n ∈ G(P) is a usage node of the variable v ∈ V, written as USE(v, n), if and only if the value of the variable v is used as the statement fragment corresponding to node n. Output statements, assignment statements, conditional statements, loop control statements, and procedure calls are all examples of statements that are usage nodes. When the code corresponding to such statements executes, the contents of the memory location(s) associated with the variables remain unchanged.

Definition 3:

A usage node USE(v, n) is a predicate use (denoted as P-use) if and only if the statement n is a predicate statement; otherwise, USE(v, n) is a computation use (denoted C-use). The nodes corresponding to predicate uses always have an outdegree ≥ 2, and nodes corresponding to computation uses always have an outdegree ≤ 1. Definition A definition/use path with respect to a variable v (denoted du-path) is a path in PATHS(P) such that, for some v ∈ V, there are define and usage nodes DEF(v, m) and USE(v, n) such that m and n are the initial and final nodes of the path.

- Definition 4:

A definition-clear path with respect to a variable v (denoted dc-path) is a definition/use path in PATHS(P) with initial and final nodes DEF(v, m) and USE(v, n) such that no other node in the path is a defining node of v.

ii. Du-paths for Stocks

First, let us look at a simple path: the du-path for the variable stocks. We have DEF(stocks, 15) and USE(stocks, 17), so the path <15, 17> is a du-path with respect to stocks. No other defining nodes are used for stocks; therefore, this path is also definition clear.

iii. Du-paths for Locks

Two defining and two usage nodes make the locks variable more interesting: we have DEF(locks,13), DEF(locks, 19), USE(locks, 14), and USE(locks, 16). These yield four du-paths; they are shown in Figure 9.3.

p1 = <13, 14>

p2 = <13, 14, 15, 16>

p3 = <19, 20, 14>

p4 = <19, 20, 14, 15, 16>

Note: du-paths p1 and p2 refer to the priming value of locks, which is read at node 13. The locks variable has a predicate use in the while statement (node 14), and if the condition is true (as in path p2), a computation use at statement 16. The other two du-paths start near the end of the while loop and occur when the loop repeats. These paths provide the loop coverage discussed in bypass the loop, begin the loop, repeat the loop, and exit the loop. All these du-paths are definition clear.

Testing Approach for System

In this there is the three levels of testing, the system level is closest to everyday experience. We test many things: a used car before we buy it, an online network service before we subscribe, and so on. A common pattern in these familiar forms is that we evaluate a product in terms of our expectations not with respect to a specification or a standard. Consequently, the goal is not to find faults but to demonstrate correct behavior.

Because of this, we tend to approach system testing from a specification-based standpoint instead of from a code-based one. Because it is so intuitively familiar, system testing in practice tends to be less formal than it might be, and this is compounded by the reduced testing interval that usually remains before a delivery deadline. The craftsperson metaphor continues to serve us. We need a better understanding of the medium; we will view system testing in terms of threads of system-level behavior.

We begin with a new construct an Atomic System Function (ASF) and develop the thread concept, highlighting some of the practical problems of thread-based system testing. System testing is closely coupled with requirements specification; therefore, we shall use appropriate system-level models to enjoy the benefits of model-based testing. Common to all of these is the idea of "threads," so we shall see how to identify system-level threads in a variety of common models. All this leads to an orderly thread-based system testing strategy that exploits the symbiosis between specification-based and code-based testing.

i. Threads:

Threads are hard to define; in fact, some published definitions are counterproductive, misleading, or wrong. It is possible to simply treat threads as a primitive concept that needs no formal definition. For now, we will use examples to develop a "shared vision." Here are several views of a thread:

1. A scenario of normal usage,
2. A system-level test case,
3. A stimulus/response pair,
4. Behavior that results from a sequence of system-level inputs
5. An interleaved sequence of port input and output events
6. A sequence of transitions in a state machine description of the system
7. An interleaved sequence of object messages and method executions
8. A sequence of machine instructions

9. A sequence of source instructions

10. A sequence of MM-paths

11. A sequence of ASFs

Basis Concepts for Requirements Specification

Recall the notion of a basis of a vector space: a set of independent elements from which all the elements in the space can be generated. Instead of anticipating all the variations in scores of requirements specification methods, notations, and techniques, we will discuss system testing with respect to a basis set of requirements specification constructs: data, actions, devices, events, and threads. Every system can be modeled in terms of these five fundamental concepts (and every requirements specification model uses some combination of these). We examine these fundamental concepts here to see how they support the tester's process of thread identification.

ii. Data

When a system is described in terms of its data, the focus is on the information used and created by the system. We describe data in terms of variables, data structures, fields, records, data stores, and files. Entity/relationship (E/R) models are the most common choice at the highest level, and some form of a regular expression (e.g., Jackson diagrams or data structure diagrams) is used at a more detailed level. The data-centered view is also the starting point for several flavors of object-oriented analysis. Data refers to information that is either initialized, stored, updated, or destroyed. In the SATM system, initial data describes the various accounts (each with its Personal Account Number, or PAN) and their PINs, and each account has a data structure with information such as the account balance. As ATM transactions occur, the results are kept as created data and used in the daily posting of terminal data to the central bank. For many systems, the data-centered view dominates. These systems are often developed in terms of CRUD actions (Create, Retrieve, Update, and Delete).

We could describe the transaction portion of the SATM system in this way, but it would not work well for the user interface portion. Sometimes threads can be identified directly from the data model. Relationships among data entities can be one-to-one, one-to-many, many-to-one, or many-to-many; these distinctions all have implications for threads that process the data. For example, if bank customers can have several accounts, each account needs a unique PIN. If several people can access the same account, they need ATM cards with identical PANs. We can also find initial data such as PAN, Expected-PIN pairs that are read but never written. Such read-only data must be part of the system initialization process. If not, there must be threads that create such data. Read-only data is therefore an indicator of source ASFs.

iii. Actions:

Action-centered modeling is still a common requirements specification form. This is a historical outgrowth of the action-centered nature of imperative programming languages. Actions have inputs and outputs, and these can be either data or port events. Here are some methodology-specific synonyms for actions: transform, data transform, control transform, process, activity, task, method, and service. Actions can also be decomposed into lower-level actions, most notably in the data flow diagrams of Structured Analysis. The input/output view of actions is exactly the basis of specification-based testing, and the decomposition (and eventual implementation) of actions is the basis of code-based testing.

iv. Devices:

Every system has port devices; these are the sources and destinations of system-level inputs and outputs. The slight distinction between ports and port devices is sometimes helpful to testers. Technically, a port is the point at which an I/O device is attached to a system, as in serial and parallel ports, network ports, and telephone ports. Physical actions occur on port devices, and these are translated from physical to logical forms. In the absence of actual port devices, much of system testing can be accomplished by "moving the port boundary inward" to the logical instances of port events. From now on, we will just use the term "port" to refer to port devices. The ports in the SATM system include the digit and cancel keys, the function keys, the display screen, the deposit and withdrawal doors, the card and receipt slots, and several less obvious devices, such as the rollers that move cards and deposit envelopes into the machine, the cash dispenser, the receipt printer, and so on and these process are mention in below Figure 6.5.

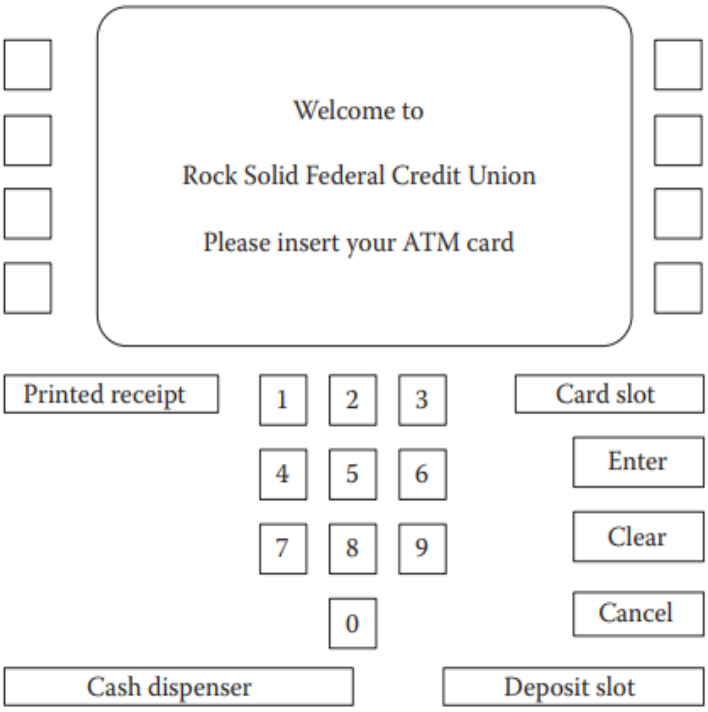


Figure 6.5: Displayed that the Simple ATM (SATM) Terminal.

v. Events:

Events are somewhat schizophrenic: they have some characteristics of data and some of actions. An event is a system-level input (or output) that occurs on a port device. Similar to data, events can be inputs to or outputs of actions. Events can be discrete (such as SATM keystrokes) or they can be continuous (such as temperature, altitude, or pressure). Discrete events necessarily have a time duration, and this can be a critical factor in real-time systems. We might picture input events as destructive read-out data, but it is a stretch to imagine output events as destructive write operations. Events are like actions in the sense that they are the translation point between real-world physical events and internal logical manifestations of these. Port input events are physical-to-logical translations, and, symmetrically, port output events are logical-to-physical translations. System testers should focus on the physical side of events, not the logical side (the focus of

integration testers). Situations occur where the context of present data values changes the logical meaning of physical events. In the SATM system, for example, the port input event of depressing button B1 means "balance" when screen 5 is displayed, "checking" when screen 6 is displayed, and "yes" when screens 10, 11, and 14 are displayed. We refer to these situations as "context-sensitive port events," and we would expect to test such events in each context.

vi. Threads:

Unfortunately for testers, threads are the least frequently used of the five fundamental constructs. Because we test threads, it usually falls to the tester to find them in the interactions among the data, events, and actions. About the only place that threads appear per se in a requirements specification is when rapid prototyping is used in conjunction with a scenario recorder. It is easy to find threads in control models, as we will soon see. The problem with this is that control models are just that they are models, not the reality of a system.

vii. Relationships among Basis Concepts:

E/R model of our basis concepts as mention in the Figure 6.6. Notice that all relationships are many-to-many: Data and Events are inputs to or outputs of the Action entity. The same event can occur on several ports, and typically many events occur on a single port. Finally, an action can occur in several threads, and a thread is composed of several actions. This diagram demonstrates some of the difficulty of system testing. Testers must use events and threads to ensure that all the many-to-many relationships among the five basis concepts are correct.

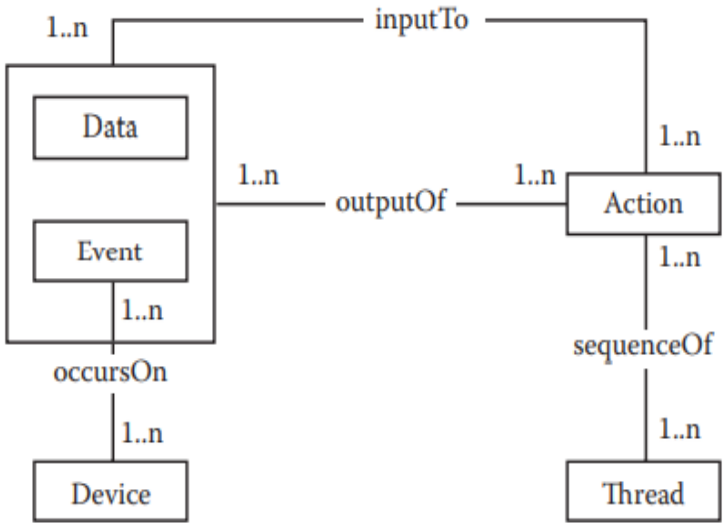


Figure 6.6: Displayed that the E/R model of basic Concepts.

viii. Model-Based Threads:

In this section, we will use the SATM system to illustrate how threads can be identified from models. Finite state machine models of the SATM system are the best place to look for system testing threads. We will start with a hierarchy of state machines; the upper level is shown in Figure 6.7. At this level, states correspond to stages of processing, and transitions are caused by abstract logical events. The card entry "state," for example, would be decomposed into lower levels that deal with details such as jammed cards, cards that are upside down, stuck card rollers, and checking

the card against the list of cards for which service is offered. Once the details of a macro-state are tested, we use a simple thread to get to the next macro-state.

The adjacent states are shown because they are sources and destinations of transitions from the PIN entry state at the upper level. This approach to decomposition is reminiscent of the old data flow diagramming idea of balanced decomposition. At the S2-decomposition, we focus on the PIN retry mechanism; all of the output events are true port events, but the input events are still logical events. In that finite state machine, we still have abstract input events, but the output events are actual port events. Two steps are combined into this state: choice of the account type and selection of the transaction type. The little "" symbols are supposed to point to the function buttons adjacent to the screen. As a side note, if this were split into two states, the system would have to "remember" the account type choice in the first state. However, there can be no memory in a finite state machine, hence the combined state. Once again, we have abstract input events and true port output events.

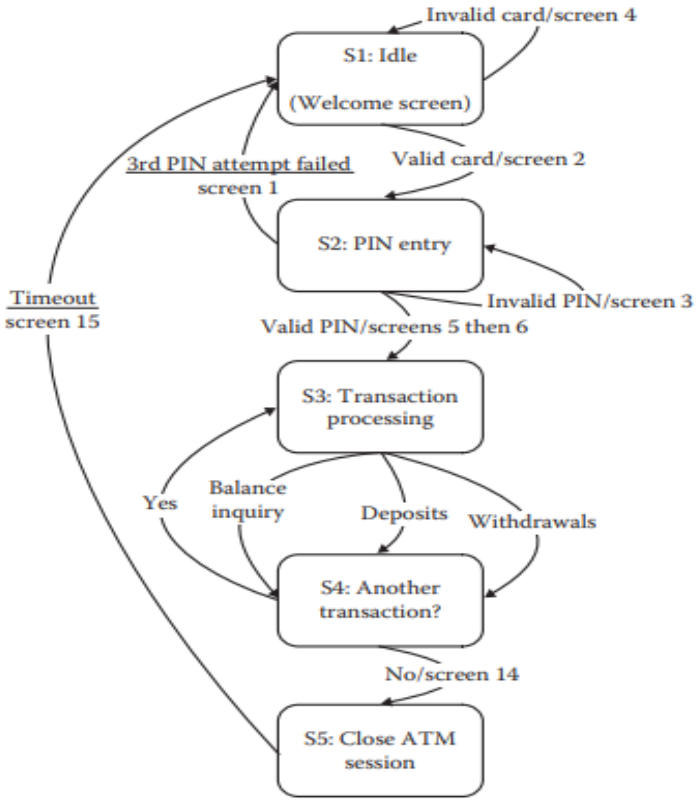


Figure 6.7: Displayed the Uppermost level SATM finite State Machine.

ix. Levels of Use Cases

One author defines a hierarchy of use cases in which each level adds information to the predecessor level. Larman names these levels as follows:

- High level (very similar to an agile user story)
- Essential
- Expanded essential

- Real

The information content of these variations is shown in Venn diagram form in Figure 28. High-level use cases are at the level of the user stories used in agile development. A set of high-level use cases gives a quick overview of the view of a system as mention in Figure 6.8. Essential use cases add the sequence of port input and output events. At this stage, the port boundary begins to become clear to both the customer/user and the developer. Expanded essential use cases add pre- and post-conditions. We shall see that these are key to linking use cases when they are expressed as system test cases. Real use cases are at the actual system test case level. Abstract names for port events, such as "invalid PIN," are replaced by an actual invalid PIN character string as display in below figure. This presumes that some form of testing database has been assembled. In our SATM system, this would likely include several accounts with associated PINs and account balances.

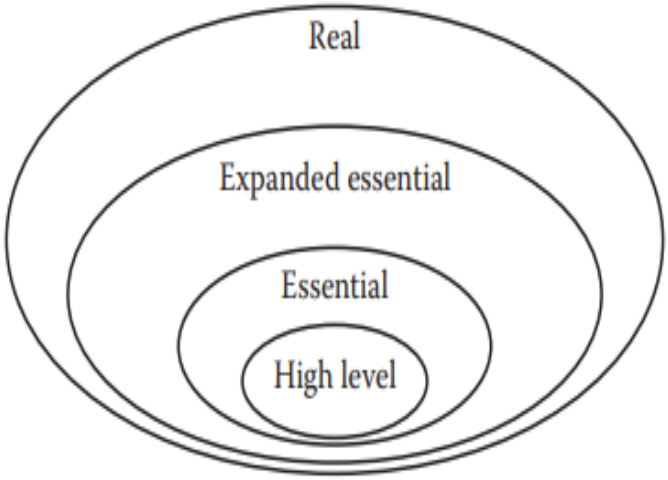


Figure 6.8: Represented that the Different Levels of Use Cases.

Related Question for Practice

1. What is Software Testing?
2. What is accessibility testing?
3. What is Adhoc testing?
4. What is Agile testing?
5. What do you mean by automated testing?
6. Which types are testing are important for web testing?
7. What is the difference between verification and validation?
8. What is the difference between Retesting and Regression Testing?
9. What are the types of performance testing?
10. What is the difference between functional and non-functional testing?
11. What are the different models available in SDLC?
12. Why do we need to perform compatibility testing?

Reference of Book for Further Reading

1. Pradeep Soundarajan “Buddha in Testing: Finding Peace in Chaos”
2. Glenford J. Myers “The Art of Software Testing”
3. Ron Patton “Software Testing”
4. Georgia Weidman “Penetration Testing – A Hands-On Introduction to Hacking”
5. Paul C. Jorgensen “Software Testing: A Craftsman’s Approach”
6. Gojko Adzic “Software Testing: A Craftsman’s Approach”
7. Dorothy Graham, Rex Black, Erik van Veenendaal “Foundations of Software Testing”
8. Cem Kaner, James Bach, Bret Pettichord “Lessons Learned in Software Testing: A Context-Driven Approach”
9. Jonathan Rasmusson “The Way of the Web Tester: A Beginner’s Guide to Automating Tests”
10. Alan J Richardson “Dear Evil Tester: Provocative Advice That Could Change Your Approach To Testing Forever”
11. Crispin Lisa, Gregory Janet “Agile Testing: A Practical Guide for Testers and Agile Teams”
12. Lee Copeland “A Practitioner's Guide to Software Test Design (Artech House Computing Library)”
13. Adam Goucher, Tim Riley “Beautiful Testing: Leading Professionals Reveal How They Improve Software”
