THEMATIC CYCLE 1:

SOFTWARE TESTING

TABLE OF CONTENTS

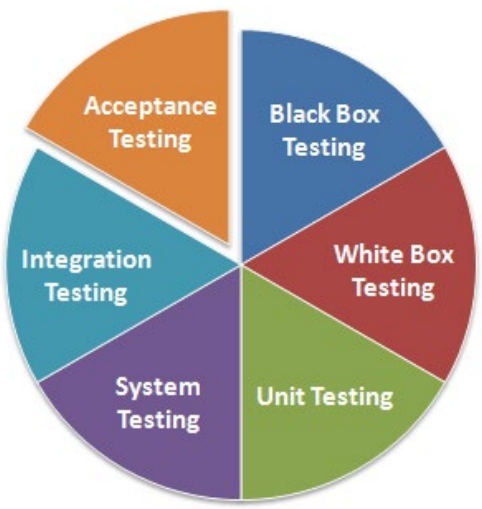
1. GOAL OF MANUAL TESTING

Manual Testing is a process carried out to find defects. In this method the tester plays an important role as end user and verifies all features of the application to ensure the behavior of the application. The Manual Testing is very basic type of testing which helps to find bugs in the application under test.

It is preliminary testing, must be carried out prior to start automating the test cases and also needs to check the feasibility of automation testing. The Test Plan is created & followed by the tester to ensure the comprehensiveness of testing while executing the test cases manually without using automation testing tool.

The main Goal of Manual Testing is to make sure that the application under test is defect free and software application is working as per the requirement specification document.

There are different stages for Manual Testing like Unit testing, Integration testing, System testing and User Acceptance testing, etc.



To start the testing process a Test Plan document must be created by test lead which describes the detailed and systematic approach to testing a software application. Basically, the Test Plan typically includes a complete understanding of what the ultimate workflow will be. To ensure the completeness of testing (100% test coverage) Test Cases or Test Scenarios are created. Manual Testing Concepts also include Exploratory testing as testers explore the software to identify errors in it.

After the testing is started the designed test cases or test scenarios are executed and any differences between actual & expected results are reported as defects. Once the reported defects are fixed, the testers will retest the defects to make sure they are fixed.

2. INTRODUCTION TO SOFTWARE TESTING AND SOFTWARE DEVELOPMENT

WHAT IS SOFTWARE TESTING:

Software testing is a process, to evaluate the functionality of a software application with an intent to find whether the developed software met the specified requirements or not and to identify the defects to ensure that the product is defect-free in order to produce the quality product.

Definition: Software Testing Definition according to ANSI/IEEE 1059 standard – A process of analyzing a software item to detect the differences between existing and required conditions (i.e., defects) and to evaluate the features of the software item.

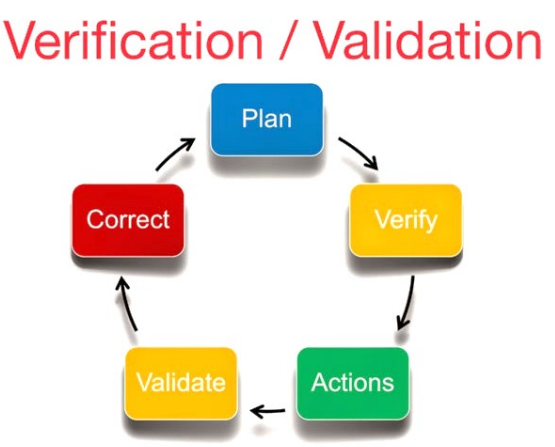
SOFTWARE TESTING TYPES:

Manual Testing: Manual testing is the process of testing software by hand to learn more about it, to find what is and isn't working. This usually includes verifying all the features specified in requirements documents, but often also includes the testers trying the software with the perspective of their end user's in mind. Manual test plans vary from fully scripted test cases, giving testers detailed steps and expected results, through to high-level guides that steer exploratory testing sessions. There are lots of sophisticated tools on the market to help with manual testing, but if you want a simple and flexible place to start, take a look at "Testpad" online.

Automation Testing: Automation testing is the process of testing the software using an automation tool to find the defects. In this process, testers execute the test scripts and generate the test results automatically by using automation tools. Some of the famous automation testing tools for functional testing are QTP/UFT and Selenium.

TESTING METHODS:

Static Testing: It is also known as Verification in Software Testing. Verification is a static method of checking documents and files. Verification is the process, to ensure whether we are building the product right i.e., to verify the requirements which we have and to verify whether we are developing the product accordingly or not.



Activities involved here are Inspections, Reviews, Walkthroughs.

Dynamic Testing: It is also known as Validation in Software Testing. Validation is a dynamic process of testing the real product. Validation is the process, whether we are building the right product i.e., to validate the product which we have developed is right or not.

Activities involved in this is Testing the software application.

TESTING APPROACHES:

There are three main types of software testing approaches:

1. White Box Testing
2. Black Box Testing

3. Grey Box Testing

White Box Testing: It is also called as Glass Box, Clear Box, Structural Testing. White Box Testing is based on application's internal code structure. In white-box testing, an internal perspective of the system, as well as programming skills, are used to design test cases. This testing is usually done at the unit level.

Black Box Testing: It is also called as Behavioral/Specification-Based/Input-Output Testing. Black Box Testing is a software testing method in which testers evaluate the functionality of the software under test without looking at the internal code structure.

Grey Box Testing: Grey box is the combination of both White Box and Black Box Testing. The tester who works on this type of testing needs to have access to design documents. This helps to create better test cases in this process.

TESTING LEVELS:

1. Unit Testing
2. Integration Testing
3. System Testing
4. Acceptance Testing

Unit Testing: Unit Testing is done to check whether the individual modules of the source code are working properly. i.e. testing each and every unit of the application separately by the developer in the developer's environment. It is AKA Module Testing or Component Testing.

Integration Testing: Integration Testing is the process of testing the connectivity or data transfer between a couple of Unit tested modules. It is AKA I&T Testing or String Testing. It is subdivided into the Top-Down Approach, Bottom-Up Approach, and Sandwich Approach (Combination of Top-Down and Bottom-Up).

System Testing (end to end testing): It's a black box testing. Testing the fully integrated application this is also called as an end to end scenario testing. To ensure that the software works in all intended target systems. Verify thorough testing of every input in the application to check for desired outputs. Testing of the user's experiences with the application.

Acceptance Testing: To obtain customer sign-off so that software can be delivered and payments received. Types of Acceptance Testing are Alpha, Beta & Gamma Testing.

TYPES OF BLACK BOX TESTING:

1. Functionality Testing
2. Non-functionality Testing

Functional testing: In simple words, what the system actually does is functional testing. To verify that each function of the software application behaves as specified in the requirement document. Testing all the functionalities by providing appropriate input to verify whether the actual output is matching the expected output or not. It falls within the scope of black-box testing and the testers need not concern about the source code of the application.

Non-functional testing: In simple words, how well the system performs is non-functionality testing. Non-functional testing refers to various aspects of the software such as performance, load, stress, scalability, security, compatibility, etc., The Main focus is to improve the user experience on how fast the system responds to a request.

TESTING ARTIFACTS:

Test Artifacts are the deliverables that are given to the stakeholders of a software project. A software project which follows SDLC (Software development life cycle) undergoes the different phases before delivering to the customer. In this process, there will be some deliverables in every phase. Some of the deliverables are provided before the testing phase commences and some are provided during the testing phase and rest after the testing phase is completed.

Some of the test deliverables are as follows:

Test plan	Test script
Traceability matrix	Test suite
Test case	Test data or Test Fixture
	Test harness

Why do we need Software Testing? Why is testing required? What if there is no Software Testing in the Software Development process?

As per the current trend, due to constant change and development in digitization, our lives are improving in all areas. The way we work is also changed. We access our bank online, we

do shopping online, we order food online, and many more. We rely on software and systems. What if these systems turn out to be defective? We all know that one small bug shows a huge impact on business in terms of financial loss and goodwill. To deliver a quality product, we need to have Software Testing in the Software Development Process.

Some of the reasons why software testing becomes a very significant and integral part in the field of information technology are as follows.

1. Cost-effectiveness
2. Customer Satisfaction
3. Security
4. Product Quality

1. Cost-effectiveness: As a matter of fact, design defects can never be completely ruled out for any complex system. It is not because developers are careless but because the complexity of a system is intractable. If the design issues go undetected, then it will become more difficult to trace back defects and rectify it. It will become more expensive to fix it. Sometimes, while fixing one bug we may introduce another one in some other module unknowingly. If the bugs can be identified in the early stages of development then it costs much less to fix them. That is why it is important to find defects in the early stages of the software development life cycle. One of the benefits of testing is cost-effectiveness.

It is better to start testing earlier and introduce it in every phase of the software development life cycle and regular testing is needed to ensure that the application is developed as per the requirement.

2. Customer Satisfaction: In any business, the ultimate goal is to give the best customer satisfaction. Yes, customer satisfaction is very important. Software testing improves the user experience of an application and gives satisfaction to the customers. Happy customers mean more revenue for a business. One of the reasons why software testing is necessary is to provide the best user experience.

3. Security: This is probably the most sensitive and vulnerable part of software testing. Testing (penetration testing & security testing) helps in product security. Hackers gain unauthorized access to data. These hackers steal user information and use it for their benefit. If your product is not secured, users won't prefer your product. Users always look for trusted products. Testing helps in removing vulnerabilities in the product.

4. Product Quality: Software Testing is an art that helps in strengthening the market reputation of a company by delivering the quality product to the client as mentioned in the requirement specification documents.

Due to these reasons, software testing becomes a very significant and integral part of the Software Development process or life cycle.

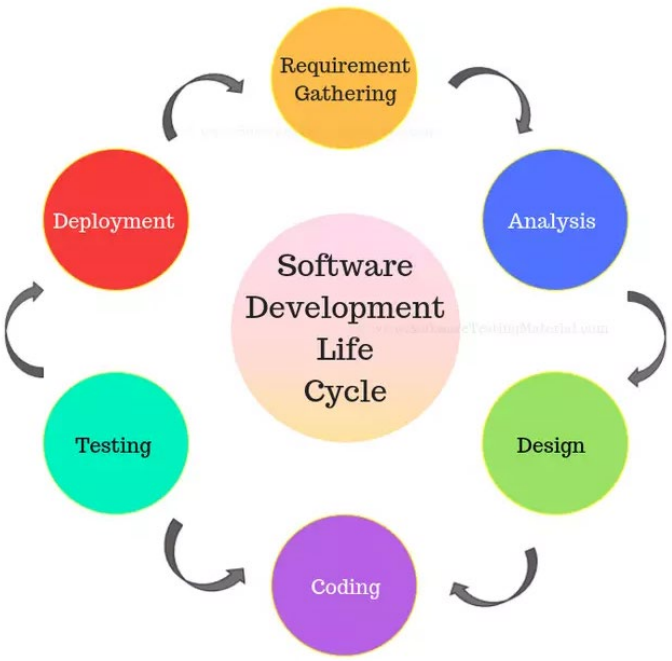
WHAT IS SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC)?

Software Development Life Cycle (SDLC) aims to produce a high-quality system that meets or exceeds customer expectations, works effectively and efficiently in the current and planned information technology infrastructure, and is inexpensive to maintain and cost-effective to enhance.

Detailed Explanation: A process followed in software projects is SDLC. Each phase of SDLC produces deliverables required by the next phase in the life cycle. Requirements are translated into design. Code is produced according to the design. Testing should be done on a developed product based on the requirement. The deployment should be done once the testing was completed. It aims to produce a high-quality system that meets or exceeds customer expectations, works effectively and efficiently in the current and planned information technology infrastructure, and is inexpensive to maintain and cost-effective to enhance.

SDLC Process: SDLC is a process which follows in Software Projects to develop a product in a systematic way and to deliver a high-quality product. By following proper SDLC process, Software companies can react well to the market pressure and release high-quality software. This process involves different stages of SDLC right from the requirement stage to deployment and maintenance phase.

A TYPICAL SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC) CONSISTS OF THE FOLLOWING PHASES:



Requirement Phase: Requirement gathering and analysis is the most important phase in the software development lifecycle. Business Analyst collects the requirement from the Customer/Client as per the clients business needs and documents the requirements in the Business Requirement Specification (document name varies depends upon the Organization. Some examples are Customer Requirement Specification (CRS), Business Specification (BS), etc., and provides the same to Development Team.

Analysis Phase: Once the requirement gathering and analysis is done the next step is to define and document the product requirements and get them approved by the customer. This is done through the SRS (Software Requirement Specification) document. SRS consists of all the product requirements to be designed and developed during the project life cycle. Key people involved in this phase are Project Manager, Business Analyst and Senior

members of the Team. The outcome of this phase is the Software Requirement Specification.

Design Phase: It has two steps

1) HLD – High-Level Design – It gives the architecture of the software product to be developed and is done by architects and senior developers
2) LLD – Low-Level Design – It is done by senior developers. It describes how each and every feature in the product should work and how every component should work. Here, only the design will be there and not the code

The outcome from this phase is High-Level Document and Low-Level Document which works as an input to the next phase.

Development Phase: Developers of all levels (seniors, juniors, freshers) involved in this phase. This is the phase where we start building the software and start writing the code for the product. The outcome from this phase is Source Code Document (SCD) and the developed product.

Testing Phase: When the software is ready, it is sent to the testing department where Test team tests it thoroughly for different defects. They either test the software manually or using automated testing tools depends on the process defined in STLC (Software Testing Life Cycle) and ensure that each and every component of the software works fine. Once the QA makes sure that the software is error-free, it goes to the next stage, which is Implementation. The outcome of this phase is the Quality Product and the Testing Artifacts.

Deployment & Maintenance Phase: After successful testing, the product is delivered/deployed to the customer for their use. Deployment is done by the Deployment/Implementation engineers. Once when the customers start using the developed system then the actual problems will come up and needs to be solved from time to time. Fixing the issues found by the customer comes in the maintenance phase. 100% testing is not possible – because, the way testers test the product is different from the way customers use the product. Maintenance should be done as per SLA (Service Level Agreement)

TYPES OF SOFTWARE DEVELOPMENT LIFE CYCLE MODELS:

There are various Software Development Life Cycle models in the industry which are followed during the software development process. These models are also referred to as Software Development Process Models.

Each SDLC model might have a different approach but the Software Development Phases and activities remain the same in all models.

Some of the Software Development Life Cycle Models (SDLC Models) followed in the industry are as follows:

1. **Waterfall Model:** Waterfall Model is a traditional model. It is aka Sequential Design Process, often used in SDLC, in which the progress is seen as flowing downwards like a waterfall, through the different phases such as Requirement Gathering, Feasibility Study/Analysis, Design, Coding, Testing, Installation, and Maintenance. Every next phase is begun only once the goal of the previous phase is completed. This methodology is preferred in projects where quality is more important as compared to schedule or cost. This methodology is best suitable for short term projects where the requirements will not change. (E.g. Calculator, Attendance Management)

2. **Spiral:** Spiral model works in an iterative nature. It is a combination of both Prototype development process and Linear development process (waterfall model). This model place more emphasis on risk analysis. Mostly this model is adopted to the large and complicated projects where risk is high. Every Iteration starts with planning and ends with the product evaluation by client.

3. **V Model:** V-model is also known as Verification and Validation (V&V) model. In this, each phase of SDLC must be completed before the next phase starts. It follows a sequential design process same like waterfall model.

4. **Prototype:** The Prototype Model is one of the mostly used Software Development Life Cycle Models (SDLC models). A prototype of the end product is first developed prior to the actual product. Usually this SDLC model is used when the customers don't know the project requirements beforehand. By developing the prototype of the end product, it gives the customers an opportunity to see the product early in the life cycle.

It starts by getting the inputs (requirements) from the customers and undergoes developing the prototype. By getting the customers feedback, requirements are refined. Actual product development starts once the customer approves the prototype. The developed product is released for customer's feedback. Released product is refined as per the customers. This process goes on until the model is accepted by the customer.

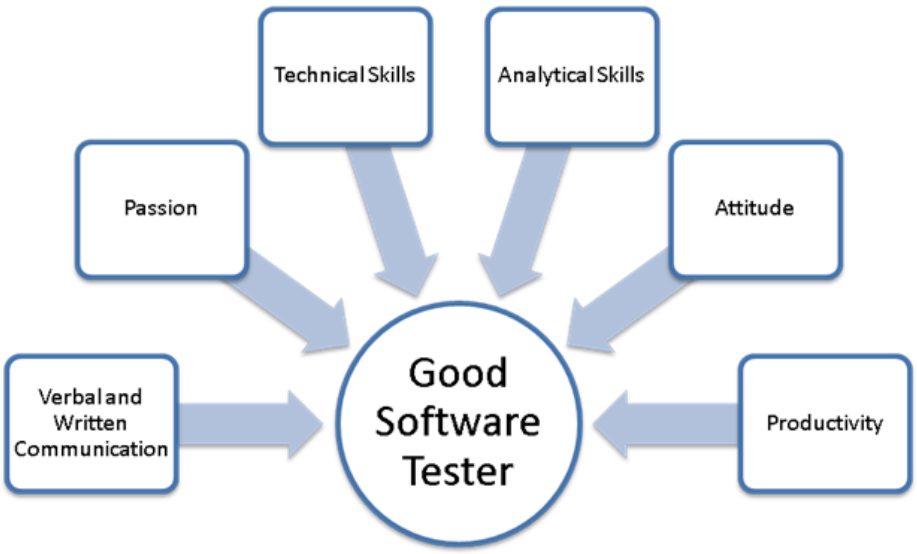
5. **Agile:** Agile Scrum Methodology is one of the popular Agile software development methods. There are some other Agile software development methods but the popular one which is used widely is Agile Scrum Methodology. The Agile Scrum Methodology is a combination of both Incremental and Iterative model for managing product development.

OTHER RELATED SOFTWARE DEVELOPMENT LIFECYCLE MODELS ARE:

Agile Model, Rapid Application Development, Rational Unified Model, Hybrid Model etc.

3. SOFTWARE TESTING AS A CAREER PATH (SKILLS, SALARY, GROWTH)

Skills required to become a Software Tester



Non-Technical Skills

Following skills are essential to becoming a good software tester. Compare your skill set against the following checklist to determine whether Software Testing is a reality for you-

Analytical skills: A good software tester should have sharp analytical skills. Analytical skills will help break up a complex software system into smaller units to gain a better understanding and create test cases. Not sure that you have good analytical skills - Refer this link - if, if you can solve at least ONE problem you have excellent analytical skills.

Communication skill: A good software tester must have good verbal and written communication skill. Testing artifacts (like test cases/plans, test strategies, bug reports, etc.) created by the software tester should be easy to read and comprehend. Dealing with developers (in the event of bugs or any other issue) will require a shade of discreetness and diplomacy.

Time Management & Organization Skills: Testing at times could be a demanding job especially during the release of code. A software tester must efficiently manage workload, have high productivity, exhibit optimal time management, and organization skills

GREAT Attitude: To be a good software tester you must have a GREAT attitude. An attitude to 'test to break', detail orientation, willingness to learn and suggest process improvements. In the software industry, technologies evolve with an overwhelming speed, and a good software tester should upgrade his/her technical skills with the changing

technologies. Your attitude must reflect a certain degree of independence where you take ownership of the task allocated and complete it without much direct supervision.

Passion: To Excel in any profession or job, one must have a significant degree of the passion for it. A software tester must have a passion for his / her field. BUT how do you determine whether you have a passion for software testing if you have never tested before? Simple TRY it out and if software testing does not excite you switch to something else that holds your interest

Technical Skills

Basic knowledge of Database/ SQL: Software Systems have a large amount of data in the background. This data is stored in different types of databases like Oracle, MySQL, etc. in the backend. So, there will be situations when this data needs to be validated. In that case, simple/complex SQL queries can be used to check whether proper data is stored in the backend databases.

Basic knowledge of Linux commands: Most of the software applications like Web-Services, Databases, Application Servers are deployed on Linux machines.So it is crucial for testers to have knowledge about Linux commands.

Knowledge and hands-on experience of a Test Management Tool:Test Management is an important aspect of Software testing. Without proper test management techniques, software testing process will fail. Test management is nothing but managing your testing related artifacts.

FOR EXAMPLE - A tool like Testlink can be used for tracking all the test cases written by your team.

There are other tools available that can be utilized for Test Management. So, it is important to have knowledge and working experience of such tools because they are used in most of the companies.

Knowledge and hands-on experience of any Defect Tracking tool- Defect Tracking and Defect life cycle are key aspects of software testing. It is extremely critical to managing defects properly and track them in a systematic manner. Defect tracking becomes necessary because the entire team should know about the defect including managers, developers, and testers. Several tools are used to lock defects including QC, Bugzilla, Jira, etc.

Knowledge and hands-on experience of Automation tool: If you see yourself as an "Automation tester" after a couple of years working on manual testing, then you must master a tool and get in-depth, hands-on knowledge of automation tools.

Note! Only knowledge of any Automation tool is not sufficient to crack the interview, you must have good hands-on experience, so practice the tool of your choice to achieve mastery.

Knowledge of any scripting language like VBScript, JavaScript, C# is always helpful as a tester if you are looking for a job into automation. Few companies also use Shell/Perl

scripting, and there is a lot of demand for testers having knowledge of the same. Again, it will depend on the company and which tools are used by that company.

Please note you do not need ALL the technical skills listed above. The technical skill sets required vary with the Job Role and company processes.

Academic Background

Academic background of a software tester should be in Computer Science. A BTech/ B.E., MCA, BCA, BSc- Computers, will land you a job quickly.

If you do not hold any of these degrees, then you must complete a software testing certification like ISTQB and CSTE which help you learn Software Development/ Test Life Cycle and other testing methodologies.

Remuneration

Compensation of a software tester varies from company to company. Average salary range of a software tester in the US is $45,993 - $74,935. Average salary range of a software tester in India is Rs 247,315 - Rs 449,111.

What Does a Software Tester do?

On any typical work day, you will be busy understanding requirement documents, creating test cases, executing test cases, reporting and re-testing bugs, attending review meetings and other team building activities.

Software Tester Career Path

Your career progression as a software tester (QA Analyst) in typical CMMI level 5 company will look like following but will vary from company to company:

1. QA Analyst (Fresher)
2. Sr. QA Analyst (2-3 years' experience)
3. QA Team Coordinator (5-6 years' experience)
4. Test Manager (8-11 years' experience)
5. Senior Test Manager (14+ experience)
6. Alternate Career Tracks as a Software Tester

Once you have got your hands dirty in manual testing, you can pursue following specializations:

Automation Testing: As an Automation Test Engineer, you will be responsible for automating manual test case execution which otherwise could be time-consuming. Tools used IBM Rational Robot, Silk performer, and QTP

Performance Testing: As a performance test engineer, you will be responsible for checking application responsiveness (time is taken to load, maximum load application can handle), etc. Tools used WEBLoad, Loadrunner.

Business Analyst: A major advantages Testers have over Developers is that they have an end to end business knowledge. An obvious career progression for testers is to become a Business Analyst. As a Business Analyst, you will be responsible for analyzing and assessing your company's business model and workflows. As a BA, you will intergrate these models and workflows with technology.

How to Become Software Tester

You start with learning Basic principles of Software Testing. Once done you apply for freelancing jobs. This will help you gain practical knowledge and will fortify the testing concepts you have learned.

Next, you proceed to Selenium - Automation tool, then JMeter - Performance Testing tool and finally TestLink - Test Management Tool. All the while you are learning, we suggest you apply for freelancing jobs (apart from other benefits you will make some money too!).

Once you are through with all the tools, you may consider taking a certification.(ISTQB) However, this is optional.

4. SOFTWARE TESTING PRINCIPLES: LEARN WITH EXAMPLES

It is important that you achieve optimum test results while conducting software testing without deviating from the goal. But how you determine that you are following the right strategy for testing? For that, you need to stick to some basic testing principles. Here are the common seven testing principles that are widely practiced in the software industry.

To understand this, consider a scenario where you are moving a file from folder A to Folder B.

Think of all the possible ways you can test this. Apart from the usual scenarios, you can also test the following conditions: Trying to move the file when it is Open; You do not have the security rights to paste the file in Folder B; Folder B is on a shared drive and storage capacity is full; Folder B already has a file with the same name, in fact, the list is endless

Or suppose you have 15 input fields to test, each having 5 possible values, the number of combinations to be tested would be 5^15

If you were to test the entire possible combinations project EXECUTION TIME & COSTS would rise exponentially. We need certain principles and strategies to optimize the testing effort.

THE 7 BASIC PRINCIPLES OF SOFTWARE TESTING:

1) EXHAUSTIVE TESTING IS NOT POSSIBLE: Yes! Exhaustive testing is not possible. Instead, we need the optimal amount of testing based on the risk assessment of the application.

For Example: If suppose we have an input field which accepts alphabets, special characters, and numbers from 0 to 1000 only. Imagine how many combinations would appear for testing, it is not possible to test all combinations for each input type.

The testing efforts required to test will be huge and it will also impact the project timeline and cost. Hence it is always said that exhaustive testing is practically not possible.

And how do you determine this risk? To answer this let's do an exercise:

In your opinion: Which operation is most likely to cause your Operating system to fail?

I am sure most of you would have guessed, Opening 10 different application all at the same time.

So, if you were testing this Operating system, you would realize that defects are likely to be found in multi-tasking activity and need to be tested thoroughly which brings us to our next principle:

2) DEFECT CLUSTERING: Defect Clustering which states that a small number of modules contain most of the defects detected. This is the application of the Pareto Principle to software testing: approximately 80% of the problems are found in 20% of the modules.

By experience, you can identify such risky modules. But this approach has its own problems. If the same tests are repeated over and over again, eventually the same test cases will no longer find new bugs.

3) PESTICIDE PARADOX: Repetitive use of the same pesticide mix to eradicate insects during farming will over time lead to the insects developing resistance to the pesticide Thereby ineffective of pesticides on insects. The same applies to software testing. If the same set of repetitive tests are conducted, the method will be useless for discovering new defects.

To overcome this, the test cases need to be regularly reviewed & revised, adding new & different test cases to help find more defects.

Testers cannot simply depend on existing test techniques. He must look out continually to improve the existing methods to make testing more effective. But even after all this sweat & hard work in testing, you can never claim your product is bug-free.

For example: Near its completion, Windows 98 was released as Windows 98 Release Candidate on April 3, 1998,[10] which expired on December 31. This coincided with a notable press demonstration at COMDEX that month. Microsoft CEO Bill Gates was highlighting the operating system's ease of use and enhanced support for Plug and Play (PnP). However,

when presentation assistant Chris Capossela plugged a USB scanner in, the operating system crashed, displaying a Blue Screen of Death. Bill Gates remarked after derisive applause and cheering from the audience, "That must be why we're not shipping Windows 98 yet."

You think a company like MICROSOFT would not have tested their OS thoroughly & would risk their reputation just to see their OS crashing during its public launch!

4) TESTING SHOWS A PRESENCE OF DEFECTS: Hence, testing principle states that - Testing talks about the presence of defects and don't talk about the absence of defects. i.e. Software Testing reduces the probability of undiscovered defects remaining in the software but even if no defects are found, it is not a proof of correctness.

But what if, you work extra hard, taking all precautions & make your software product 99% bug-free. And the software does not meet the needs & requirements of the clients.

This leads us to our next principle, which states that- Absence of Error

Example 1:

Consider a banking application, this application is thoroughly tested and undergoes different phases of testing like sit, uat etc. And currently no defects are identified in the system.

However, there might be a possibility that in the production environment, the actual customer tries a functionality which is rarely used in the banking system and the testers overlooked that functionality, hence no defect was found till date or the code has never been touched by developers.

Example 2:

We have seen several advertisements for soaps, toothpaste, handwash or disinfectant sprays etc on television.

Consider a handwash advertisement which says on the television that 99% germs can be removed if that specific handwash is used. This clearly proves that the product is not 100% germ-free. Thus in our testing concept, we can say that no software is defect free.

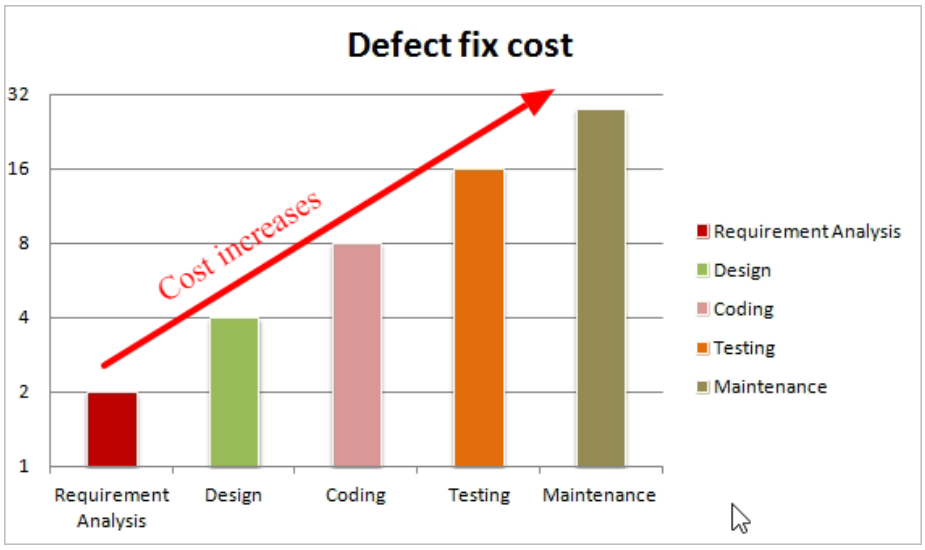
5) ABSENCE OF ERROR – FALLACY: It is possible that software which is 99% bug-free is still unusable. This can be the case if the system is tested thoroughly for the wrong requirement. Software testing is not mere finding defects, but also to check that software addresses the business needs. The absence of Error is a Fallacy i.e. Finding and fixing defects does not help if the system build is unusable and does not fulfill the user's needs & requirements.

For Example: suppose the application is related to an e-commerce site and the requirements against "Shopping Cart or Shopping Basket" functionality which is wrongly interpreted and tested. Here, even finding more defects does not help to move the application into the next phase or in the production environment.

To solve this problem, the next principle of testing states that.

6) EARLY TESTING: Early Testing - Testing should start as early as possible in the Software Development Life Cycle. So that any defects in the requirements or design phase are captured in early stages. It is much cheaper to fix a Defect in the early stages of testing. But how early one should start testing? It is recommended that you start finding the bug the moment the requirements are defined.

Consider the below image which shows how the cost of defect fixing gets increased as testing move towards the live production.



7) TESTING IS CONTEXT DEPENDENT: Testing is context dependent which basically means that the way you test an e-commerce site will be different from the way you test a commercial off the shelf application. All the developed software's are not identical. You might use a different approach, methodologies, techniques, and types of testing depending upon the application type. For instance, testing any POS system at a retail store will be different than testing an ATM machine.

There are several domains available in the market like Banking, Insurance, Medical, Travel, Advertisement etc and each domain has a number of applications. Also for each domain, their applications have different requirements, functions, different testing purpose, risk, techniques etc.

Different domains are tested differently, thus testing is purely based on the context of the domain or application.

For Example: testing a banking application is different than testing any e-commerce or advertising application. The risk associated with each type of application is different, thus it is not effective to use the same method, technique, and testing type to test all types of application.

5. MANUAL TESTING TUTORIAL FOR BEGINNERS: CONCEPTS, TYPES, TOOL

Manual Testing Definition: Manual testing a type of testing that involves validation of the requirements of the application by executing a predefined set of test cases manually without the use of any automation tool.

MANUAL TESTING PROCESS:

Requirement Understanding – In this phase, all the requirements are gathered and analyzed. This is the most important phase of testing as the requirements are the very basis of the test cases and the overall testing of the application.

Test plan and test strategy – In this phase, a document containing the scope and objective of testing is defined (Test Plan) along with deciding some principles that will define how testing will be carried out (Test Strategy).

Test case creation – After test planning and test strategy, we prepare test cases based on the functional and non-functional requirements of the application.

Test case execution and defect logging – Once test cases are ready and application is available for testing, we begin with test case execution, marking each test case as pass-fail and raising defects for each failure.

Retesting and regression – After the bug fixing by developers, we retest the bug fixes and also do regression testing, to ensure that the fixes don't adversely affect other functionalities.

Test report sharing – Once the whole test execution cycle gets completed, the test results are shared with the relevant stakeholders along with a set of known issues, if any.

ADVANTAGES OF MANUAL TESTING:

Manual testing helps in finding the defects before delivering to the customer, thus maintaining the application quality. It helps in the early identification of the bugs. Bugs found by the customer or even in the later stage of the project are difficult to fix and increase the cost of the project. Hence, efficiently carried out manual testing helps in avoiding such situations by early detection of issues. It helps in ensuring the conformance to not only the functional requirements but non-functional requirements as well as – performance, usability, and user-friendliness.

DISADVANTAGES OF MANUAL TESTING

Time-consuming – It is a very time-consuming process as testers have to create exhaustive test cases and then execute each and every step of the test cases. Also, the documentation of the test results with actual results takes time. Requires more resources – As compared to automation testing, in case of manual testing more resources are required to create and execute the test cases. Prone to human errors – The manual testing relies heavily on the ability or skills of the person creating and executing the test cases. Even with the predefined requirements and test steps, two testers can come up with different test results based on their understanding.

Not all testing can be done manually – Some of the testing like – Performance testing, security testing, or testing of scenarios like distributed testing, multi-threaded operation testing cannot be done effectively without any automation, performance or security tool.

TIPS FOR BETTER MANUAL TESTING

Get a thorough understanding of the requirements - For better manual testing, the right and complete understanding of the requirements is very vital. It helps in improving the test coverage thereby improving the overall testing.

Good Domain knowledge - Good understanding of the domain is also very important as it helps in taking care of unspecified requirements or the non-functional requirements e.g a person having knowledge of the e-commerce domain can effectively test any e-commerce application even with a limited set of requirements.

Technical skills – Having technical skills like the ability to query the database, understanding of table structure, knowledge of client-server architecture, etc helps in the

understanding of the internal working of the application and data flow. This, in turn, helps in creating better test cases.

Attention to detail – While testing, attention to detail helps in meticulously testing each and every feature of the application.

Do not assume – As testers grow in experience, at times they tend to take certain things for granted but no matter how simple or trivial the functionality to be tested is, the tester should not assume that it will work without validating the same.

Good communication skills – As a tester, one has to deal with different stakeholders, fellow developers, managers, and sometimes with the client representatives also. Good communication and interpersonal skills help testers inefficient gathering of requirements as clearly articulated queries help in avoiding any requirement gaps.

WHAT IS SOFTWARE TESTING LIFE CYCLE (STLC)

Software Testing Life Cycle (STLC) identifies what test activities to carry out and when to accomplish those test activities. Even though testing differs between Organizations, there is a testing life cycle.

The different phases of Software Testing Life Cycle are:

1. Requirement Analysis
2. Test Planning
3. Test Design
4. Test Environment Setup
5. Test Execution
6. Test Closure

Every phase of STLC (Software Testing Life Cycle) has a definite Entry and Exit Criteria.

Requirement Analysis: Entry criteria for this phase is BRS (Business Requirement Specification) document. During this phase, test team studies and analyzes the requirements from a testing perspective. This phase helps to identify whether the requirements are testable or not. If any requirement is not testable, test team can communicate with various stakeholders (Client, Business Analyst, Technical Leads, System Architects etc) during this phase so that the mitigation strategy can be planned.

Entry Criteria: BRS (Business Requirement Specification)
Deliverables: List of all testable requirements, Automation feasibility report (if applicable)

Test Planning: Test planning is the first step of the testing process. In this phase typically Test Manager/Test Lead involves determining the effort and cost estimates for the entire project. Preparation of Test Plan will be done based on the requirement analysis. Activities like resource planning, determining roles and responsibilities, tool selection (if

automation), training requirement etc., carried out in this phase. The deliverables of this phase are Test Plan & Effort estimation documents.

Entry Criteria: Requirements Documents
Deliverables: Test Strategy, Test Plan, and Test Effort estimation document.

Test Design: Test team starts with test cases development activity here in this phase. Test team prepares test cases, test scripts (if automation) and test data. Once the test cases are ready then these test cases are reviewed by peer members or team lead. Also, test team prepares the Requirement Traceability Matrix (RTM). RTM traces the requirements to the test cases that are needed to verify whether the requirements are fulfilled. The deliverables of this phase are Test Cases, Test Scripts, Test Data, Requirements Traceability Matrix

Entry Criteria: Requirements Documents (Updated version of unclear or missing requirement)
Deliverables: Test cases, Test Scripts (if automation), Test data.

Test Environment Setup: This phase can be started in parallel with Test design phase. Test environment setup is done based on the hardware and software requirement list. Some cases test team may not be involved in this phase. Development team or customer provides the test environment. Meanwhile, test team should prepare the smoke test cases to check the readiness of the given test environment.

Entry Criteria: Test Plan, Smoke Test cases, Test Data
Deliverables: Test Environment. Smoke Test Results.

Test Execution: Test team starts executing the test cases based on the planned test cases. If a test case result is Pass/Fail then the same should be updated in the test cases. Defect report should be prepared for failed test cases and should be reported to the Development Team through bug tracking tool (eg., Quality Center) for fixing the defects. Retesting will be performed once the defect was fixed.

Entry Criteria: Test Plan document, Test cases, Test data, Test Environment.
Deliverables: Test case execution report, Defect report, RTM

Test Closure: The final stage where we prepare Test Closure Report, Test Metrics. Testing team will be called out for a meeting to evaluate cycle completion criteria based on Test coverage, Quality, Time, Cost, Software, Business objectives. Test team analyses the test artifacts (such as Test cases, Defect reports etc.,) to identify strategies that have to be implemented in future, which will help to remove process bottlenecks in the upcoming projects. Test metrics and Test closure report will be prepared based on the above criteria.

Entry Criteria: Test Case Execution report (make sure there are no high severity defects opened), Defect report
Deliverables: Test Closure report, Test metrics

WHAT IS THE DIFFERENCE BETWEEN SDLC & STLC (SDLC VS STLC)? >>>

Criterion	SDLC	STLC
Origin	Development Life Cycle	Testing Life Cycle
Definition	Software Development Life Cycle (SDLC) aims to produce a high-quality system that meets or exceeds customer expectations, works effectively and efficiently in the current and planned information technology infrastructure, and is inexpensive to maintain and cost-effective to enhance.	Software Testing Life Cycle (STLC) identifies what test activities to carry out and when to accomplish those test activities. Even though testing differs between Organizations, there is a testing life cycle.
Relationship	It is taken as the predecessor	It is taken as the successor
Phases	Requirement Gathering, Analysis, Design, Coding, Testing, Deployment & maintenance	Requirement Analysis, Test Planning, Test Design, Environment Setup, Test Execution, Test Closure
Requirement Gathering Phase	Business analyst gathers the requirements and create Development Plan	QA team analyses requirement documents and create System Test Plan
Design Phase	The development team develops the high and low-level design of the software based on the requirements	Test Architect or a Test Lead usually plan the test strategy
Coding Phase	The actual code is developed as per the designed document	The QA team prepares the test environment
Testing	Actual testing is done in this phase. It includes Unit, Integration, System,	Actual testing is done in this phase. Defect reporting & retesting is done

Criterion	SDLC	STLC
Phase	Retesting & Regression testing etc., Also the development team involves in fixing the bugs reported	here
Deployment or Maintenance Phase	The development team involves in support and release updates	The QA team executes regression suites to check maintenance code deployed

WHAT IS BUG LIFE CYCLE OR DEFECT LIFE CYCLE IN SOFTWARE TESTING

Bug life cycle is also known as Defect life cycle. In Software Development process, the bug has a life cycle. The bug should go through the life cycle to be closed. Bug life cycle varies depends upon the tools (QC, JIRA etc.,) used and the process followed in the organization.

What is a Software Bug?

Software bug can be defined as the abnormal behavior of the software. Bug starts when the defect is found and ends when a defect is closed, after ensuring it is not reproduced.

The different states of a bug in the bug life cycle are as follows:

New: When a tester finds a new defect. He should provide a proper Defect document to the Development team to reproduce and fix the defect. In this state, the status of the defect posted by tester is “New”

Assigned: Defects which are in the status of New will be approved (if valid) and assigned to the development team by Test Lead/Project Lead/Project Manager. Once the defect is assigned then the status of the bug changes to “Assigned”

Open: The development team starts analyzing and works on the defect fix

Fixed: When a developer makes the necessary code change and verifies the change, then the status of the bug will be changed as “Fixed” and the bug is passed to the testing team.

Test: If the status is “Test”, it means the defect is fixed and ready to do test whether it is fixed or not.

Verified: The tester re-tests the bug after it got fixed by the developer. If there is no bug detected in the software, then the bug is fixed and the status assigned is “verified.”

Closed: After verified the fix, if the bug is no longer exits then the status of bug will be assigned as “Closed.”

Reopen: If the defect remains same after the retest, then the tester posts the defect using defect retesting document and changes the status to “Reopen”. Again the bug goes through the life cycle to be fixed.

Duplicate: If the defect is repeated twice or the defect corresponds the same concept of the bug, the status is changed to “duplicate” by the development team.

Deferred: In some cases, Project Manager/Lead may set the bug status as deferred.
If the bug found during end of release and the bug is minor or not important to fix immediately
If the bug is not related to current build
If it is expected to get fixed in the next release
Customer is thinking to change the requirement
In such cases the status will be changed as “deferred” and it will be fixed in the next release.

Rejected: If the system is working according to specifications and bug is just due to some misinterpretation (such as referring to old requirements or extra features) then Team lead or developers can mark such bugs as “Rejected”

Some other statuses are:

Cannot be fixed: Technology not supporting, Root of the product issue, Cost of fixing bug is more

Not Reproducible: Platform mismatch, improper defect document, data mismatch, build mismatch, inconsistent defects

Need more information: If a developer is unable to reproduce the bug as per the steps provided by a tester then the developer can change the status as “Need more information’. In this case, the tester needs to add detailed reproducing steps and assign bug back to the development team for a fix. This won’t happen if the tester writes a good defect document.

Bug Report Template – Detailed Explanation

Defect report template or Bug report template is one of the test artifacts. It comes into picture when the test execution phase is started.

The purpose of using Defect report template or Bug report template is to convey the detailed information (like environment details, steps to reproduce etc.,) about the bug to the developers. It allows developers to replicate the bug easily.

Components of Bug Report Template:

1) Defect ID: Add a Defect ID using a naming convention followed by your team. The Defect ID will be generated automatically in case of defect management tool.
2) Title/Summary: Title should be short and simple. It should contain specific terms related to the actual issue. Be specific while writing the title.

Example: Assume, you have found a bug in the registration page while uploading a profile picture that too a particular file format (i.e., JPEG file). System is crashing while uploading a JPEG file.

Good: "Uploading a JPEG file (Profile Picture) in the Registration Page crashes the system"

Bad: "System crashes".

3) Reporter Name: Name of the one who found the defect (Usually tester's name but sometimes it might be Developer, Business Analyst, Subject Matter Expert (SME), Customer)
4) Defect Reported Date: Mention the date on which you have found the bug.
5) Who Detected: Specify the designation of the one who found the defect. E.g. QA, Developer, Business Analyst, SME, Customer
6) How Detected: In this field, you must specify on how you have detected such as while doing Testing or while doing Review or while giving Walkthrough etc.,
7) Project Name: Sometimes, we may work on multiple projects simultaneously. So, choose the project name correctly. Specify the name of the project (If it's a product, specify the product name)
8) Release/Build Version: On which release this issue occurs. Mention the build version details clearly.
9) Defect/Enhancement: If the system is not behaving as intended then you need to specify it as a Defect. If its just a request for a new feature then you must specify it as Enhancement.
10) Environment: You must mention the details of Operation Systems, Browser Details and any other related to the test environment in which you have encountered the bug. (Example: Windows 8/Chrome 48.0.2564.103)
11) Priority: Priority defines how soon the bug should be fixed. Usually, the priority of the bug is set by the Managers. Based on the priority, developers could understand how soon it must be fixed and set the order in which a bug should be resolved.
    a) Categories of Priority:

    High Medium Low

12) Severity: Severity talks about the impact of the bug on the customer's business. Usually, the severity of the bug is set by the Managers. Sometimes, testers choose the severity of the bug but in most cases, it will be selected by Managers/Leads.

a) Categories of Severity:

Blocker Critical Major Minor Trivial

13) Status: Specify the status of the bug. If you just found a bug and about to post it then the status will be “New”. In the course of bug fixing, the status of the bug will change.

(E.g. New/ Assigned/ Open/ Fixed/ Test/ Verified/ Closed/ Reopen/ Duplicate/ Deferred/ Rejected/ cannot be fixed/ Not Reproducible/ Need more information)

14) Description: In the description section, you must briefly explain what you have done before facing the bug.

15) Steps to reproduce: In this section, you should describe how to reproduce the bug in step by step manner. Easy to follow steps give room to the developers to fix the issue without any chaos. These steps should describe the bug well enough and allows developers to understand and act on the bug without discussing to the one who wrote the bug report. Start with “opening the application”, include “prerequisites” if any and write till the step which “causes the bug”.

Example:

Good:

i. Open URL “Your URL”
ii. Click on “Registration Page”
iii. Upload “JPEG” file in the profile photo field

Bad:

Upload a file in the registration page.

16) URL: Mention the URL of the application (If available)

17) Expected Result: What is the expected output from the application when you make an action which causes failure.

Example:

Good: A message should display “Profile picture uploaded successfully”

Bad: System should accept the profile picture.

18) Actual Result: What is the expected output from the application when you make an action which causes failure.

Example:

Good: “Uploading a JPEG file (Profile Picture) in the Registration Page crashes the system”.

Bad: System is not accepting profile picture.

19) Attachments: Attach the screenshots which you had captured when you faced the bug. It helps the developers to see the bug which you have faced.

20) Defect Close Date: The 'Defect Close Date' is the date which needs to be updated once you ensure that the defect is not reproducible.

DIFFERENCE BETWEEN DEFECT, BUG, ERROR AND FAILURE

Let's see the difference between defect, bug, error and failure. In general, we use these terms whenever the system/application acts abnormally. Sometimes we call it's an error and sometimes bug and so on. Many of the newbies in Software Testing industry have confusion in using this.

Generally, there is a contradiction in the usage of these terminologies. Usually in Software Development Life Cycle we use these terms based on the phase.

Note: Both Defect and Bug are the issues in an application but in which phase of SDLC it was found makes the overall difference.

What is a defect?

The variation between the actual results and expected results is known as defect. If a developer finds an issue and corrects it by himself in the development phase then it's called a defect.

What is a bug?

If testers find any mismatch in the application/system in testing phase then they call it as Bug. As mentioned earlier, there is a contradiction in the usage of Bug and Defect. People widely say the bug is an informal name for the defect.

What is an error?

We can't compile or run a program due to coding mistake in a program. If a developer unable to successfully compile or run a program then they call it as an error.

What is a failure?

Once the product is deployed and customers find any issues then they call the product as a failure product. After release, if an end user finds an issue then that particular issue is called as failure

Points to know: If a Quality Analyst (QA) finds a bug, he has to reproduce and record it using the bug report template.

COMMON TYPES OF SOFTWARE TESTING:

Unit Testing

Integration Testing

System Testing

Sanity Testing

Smoke Testing

Interface Testing

Regression Testing

Beta/Acceptance Testing

Non-functional Testing types include:

Performance Testing

Load Testing

Stress Testing

Volume Testing

Security Testing

Compatibility Testing

Install Testing

Recovery Testing

Reliability Testing

Usability Testing

Compliance Testing

Localization Testing

Let's see more details about these Testing types.

#1) Alpha Testing: It is the most common type of testing used in the Software industry. The objective of this testing is to identify all possible issues or defects before releasing it into the market or to the user.

Alpha Testing is carried out at the end of the software development phase but before the Beta Testing. Still, minor design changes may be made as a result of such testing.

Alpha Testing is conducted at the developer's site. In-house virtual user environment can be created for this type of testing.

#2) Acceptance Testing: An Acceptance Test is performed by the client and verifies whether the end to end the flow of the system is as per the business requirements or not and if it is as per the needs of the end-user. Client accepts the software only when all the features and functionalities work as expected.

It is the last phase of the testing, after which the software goes into production. This is also called User Acceptance Testing (UAT).

#3) Ad-hoc Testing: The name itself suggests that this testing is performed on an Ad-hoc basis i.e. with no reference to the test case and also without any plan or documentation in place for such type of testing.

The objective of this testing is to find the defects and break the application by executing any flow of the application or any random functionality.

Ad-hoc Testing is an informal way of finding defects and can be performed by anyone in the project. It is difficult to identify defects without a test case but sometimes it is possible that defects found during ad-hoc testing might not have been identified using existing test cases.

#4) Accessibility Testing: The aim of Accessibility Testing is to determine whether the software or application is accessible for disabled people or not.

Here, disability means deaf, color blind, mentally disabled, blind, old age and other disabled groups. Various checks are performed such as font size for visually disabled, color and contrast for color blindness, etc.

#5) Beta Testing: Beta Testing is a formal type of Software Testing which is carried out by the customer. It is performed in the Real Environment before releasing the product to the market for the actual end-users.

Beta Testing is carried out to ensure that there are no major failures in the software or product and it satisfies the business requirements from an end-user perspective. Beta Testing is successful when the customer accepts the software. Usually, this testing is typically done by end-users or others. It is the final testing done before releasing an application for commercial purpose. Usually, the Beta version of the software or product released is limited to a certain number of users in a specific area.

So end-user actually uses the software and shares the feedback to the company. Company then takes necessary action before releasing the software to the worldwide.

#6) Back-end Testing: Whenever an input or data is entered on front-end application, it stores in the database and the testing of such database is known as Database Testing or Backend Testing.

There are different databases like SQL Server, MySQL, and Oracle, etc. Database Testing involves testing of table structure, schema, stored procedure, data structure and so on.

In Back-end Testing GUI is not involved, testers are directly connected to the database with proper access and testers can easily verify data by running a few queries on the database.

There can be issues identified like data loss, deadlock, data corruption etc during this back-end testing and these issues are critical to fixing before the system goes live into the production environment

#7) Browser Compatibility Testing: It is a subtype of Compatibility Testing (which is explained below) and is performed by the testing team. Browser Compatibility Testing is performed for web applications and it ensures that the software can run with the combination of different browser and operating system. This type of testing also validates whether web application runs on all versions of all browsers or not.

#8) Backward Compatibility Testing: It is a type of testing which validates whether the newly developed software or updated software works well with the older version of the environment or not. Backward Compatibility Testing checks whether the new version of the software works properly with file format created by an older version of the software; it also works well with data tables, data files, data structure created by the older version of that software.

If any of the software is updated then it should work well on top of the previous version of that software.

#9) Black Box Testing: Internal system design is not considered in this type of testing. Tests are based on the requirements and functionality.

#10) Boundary Value Testing: This type of testing checks the behavior of the application at the boundary level. Boundary Value Testing is performed for checking if defects exist at boundary values. Boundary Value Testing is used for testing a different range of numbers. There is an upper and lower boundary for each range and testing is performed on these boundary values.

If testing requires a test range of numbers from 1 to 500 then Boundary Value Testing is performed on values at 0, 1, 2, 499, 500 and 501.

#11) Branch Testing: It is a type of White box Testing and is carried out during Unit Testing. Branch Testing, the name itself suggests that the code is tested thoroughly by traversing at every branch.

#12) Comparison Testing: Comparison of a product's strength and weaknesses with its previous versions or other similar products is termed as Comparison Testing.

#13) Compatibility Testing: It is a testing type in which it validates how software behaves and runs in a different environment, web servers, hardware, and network environment. Compatibility testing ensures that software can run on a different configuration, different database, different browsers, and their versions. Compatibility testing is performed by the testing team.

#14) Component Testing: It is mostly performed by developers after the completion of unit testing. Component Testing involves testing of multiple functionalities as a single code and its objective is to identify if any defect exists after connecting those multiple functionalities with each other.

#15) End-to-End Testing: Similar to system testing, End-to-End Testing involves testing of a complete application environment in a situation that mimics real-world use, such as interacting with a database, using network communications, or interacting with other hardware, applications, or systems if appropriate.

#16) Equivalence Partitioning: It is a testing technique and a type of Black Box Testing. During this Equivalence Partitioning, a set of the group is selected and a few values or numbers are picked up for testing. It is understood that all values from that group generate the same output.

The aim of this testing is to remove redundant test cases within a specific group which generates the same output but not any defect.

Suppose, the application accepts values between -10 to +10 so using equivalence partitioning the values picked up for testing are zero, one positive value, one negative value. So the Equivalence Partitioning for this testing is -10 to -1, 0, and 1 to 10.

#17) Example Testing: It means real-time testing. Example Testing includes the real-time scenario, it also involves the scenarios based on the experience of the testers.

#18) Exploratory Testing: Exploratory Testing is informal testing performed by the testing team. The objective of this testing is to explore the application and looking for defects that exist in the application. Sometimes it may happen that during this testing major defect discovered can even cause a system failure. During Exploratory Testing, it is advisable to keep a track of what flow you have tested and what activity you did before the start of the specific flow.

An Exploratory Testing technique is performed without documentation and test cases.

#20) Functional Testing: This type of testing ignores the internal parts and focuses only on the output to check if it is as per the requirement or not. It is a Black-box type testing geared to the functional requirements of an application.

#21) Graphical User Interface (GUI) Testing: The objective of this GUI Testing is to validate the GUI as per the business requirement. The expected GUI of the application is mentioned in the Detailed Design Document and GUI mockup screens. The GUI Testing includes the size of the buttons and input field present on the screen, alignment of all text, tables, and content in the tables.

It also validates the menu of the application, after selecting different menu and menu items, it validates that the page does not fluctuate and the alignment remains same after hovering the mouse on the menu or sub-menu.

#22) Gorilla Testing: Gorilla Testing is a testing type performed by a tester and sometimes by the developer the as well. In Gorilla Testing, one module or the functionality in the module is tested thoroughly and heavily. The objective of this testing is to check the robustness of the application.

#23) Happy Path Testing: The objective of Happy Path Testing is to test an application successfully on a positive flow. It does not look for negative or error conditions. The focus is only on the valid and positive inputs through which application generates the expected output.

#24) Incremental Integration Testing: Incremental Integration Testing is a Bottom-up approach for testing i.e continuous testing of an application when new functionality is added. Application functionality and modules should be independent enough to test separately. This is done by programmers or by testers.

#25) Install/Uninstall Testing: Installation and Uninstallation Testing is done on full, partial, or upgrade install/uninstall processes on different operating systems under different hardware or software environment.

#26) Integration Testing: Testing of all integrated modules to verify the combined functionality after integration is termed as Integration Testing. Modules are typically code modules, individual applications, client and server applications on a network, etc. This type of testing is especially relevant to client/server and distributed systems.

#27) Load Testing: It is a type of Non-Functional Testing and the objective of Load Testing is to check how much load or maximum workload a system can handle without any performance degradation. Load Testing helps to find the maximum capacity of the system under specific load and any issues that cause software performance degradation. Load testing is performed using tools like JMeter, LoadRunner, WebLoad, Silk performer, etc.

#28) Monkey Testing: Monkey Testing is carried out by a tester assuming that if the monkey uses the application then how random input, values will be entered by the Monkey without any knowledge or understanding of the application.

The objective of Monkey Testing is to check if an application or system gets crashed by providing random input values/data. Monkey Testing is performed randomly and no test cases are scripted.

#29) Mutation Testing: Mutation Testing is a type of white box testing in which the source code of one of the program is changed and verifies whether the existing test cases can identify these defects in the system. The change in the program source code is very minimal so that it does not impact the entire application, only the specific area having the impact and the related test cases should able to identify those errors in the system.

#30) Negative Testing: Testers having the mindset of “attitude to break” and using Negative Testing they validate that if system or application breaks. A Negative Testing technique is performed using incorrect data, invalid data or input. It validates that if the system throws an error of invalid input and behaves as expected.

#31) Non-Functional Testing: It is a type of testing for which every organization having a separate team which usually called as Non-Functional Test (NFT) team or Performance team. Non-Functional Testing involves testing of non-functional requirements such as Load Testing, Stress Testing, Security, Volume, Recovery Testing, etc. The objective of NFT testing is to ensure whether the response time of software or application is quick enough as per the business requirement.

It should not take much time to load any page or system and should sustain during peak load.

#32) Performance Testing: This term is often used interchangeably with ‘stress' and ‘load' testing. Performance Testing is done to check whether the system meets the performance requirements. Different performance and load tools are used to do this testing.

#33) Recovery Testing: It is a type of testing which validates how well the application or system recovers from crashes or disasters. Recovery Testing determines if the system is

able to continue the operation after a disaster. Assume that application is receiving data through the network cable and suddenly that network cable has been unplugged. Sometime later, plug the network cable; then the system should start receiving data from where it lost the connection due to network cable unplugged.

#34) Regression Testing: Testing an application as a whole for the modification in any module or functionality is termed as Regression Testing. It is difficult to cover all the system in Regression Testing, so typically Automation Testing Tools are used for these types of testing.

#35) Risk-Based Testing (RBT): In Risk-Based Testing, the functionalities or requirements are tested based on their priority. Risk-Based Testing includes testing of highly critical functionality, which has the highest impact on business and in which the probability of failure is very high. The priority decision is based on the business need, so once priority is set for all functionalities then high priority functionality or test cases are executed first followed by medium and then low priority functionalities. The low priority functionality may be tested or not tested based on the available time.

The Risk-Based Testing is carried out if there is insufficient time available to test entire software and software needs to be implemented on time without any delay. This approach is followed only by the discussion and approval of the client and senior management of the organization.

#36) Sanity Testing: Sanity Testing is done to determine if a new software version is performing well enough to accept it for a major testing effort or not. If an application is crashing for the initial use then the system is not stable enough for further testing. Hence a build or an application is assigned to fix it.

#37) Security Testing: It is a type of testing performed by a special team of testers. A system can be penetrated by any hacking way. Security Testing is done to check how the software or application or website is secure from internal and external threats. This testing includes how much software is secure from the malicious program, viruses and how secure and strong the authorization and authentication processes are. It also checks how software behaves for any hackers attack and malicious programs and how software is maintained for data security after such a hacker attack.

#38) Smoke Testing: Whenever a new build is provided by the development team then the Software Testing team validates the build and ensures that no major issue exists. The testing team ensures that the build is stable and a detailed level of testing is carried out further. Smoke Testing checks that no show stopper defect exists in the build which will prevent the testing team to test the application in detail.

If testers find that the major critical functionality is broken down at the initial stage itself then testing team can reject the build and inform accordingly to the development team. Smoke Testing is carried out to a detailed level of any Functional or Regression Testing.

#39) Static Testing: Static Testing is a type of testing which is executed without any code. The execution is performed on the documentation during the testing phase. It involves reviews, walkthrough, and inspection of the deliverables of the project. Static Testing does not execute the code instead of the code syntax, naming conventions are checked. Static Testing is also applicable for test cases, test plan, design document. It is necessary to perform static testing by the testing team as the defects identified during this type of testing are cost-effective from the project perspective.

#40) Stress Testing: This testing is done when a system is stressed beyond its specifications in order to check how and when it fails. This is performed under heavy load like putting large number beyond storage capacity, complex database queries, continuous input to the system or database load.

#41) System Testing: Under System Testing technique, the entire system is tested as per the requirements. It is a Black-box type Testing that is based on overall requirement specifications and covers all the combined parts of a system.

#42) Unit Testing: Testing of an individual software component or module is termed as Unit Testing. It is typically done by the programmer and not by testers, as it requires detailed knowledge of the internal program design and code. It may also require developing test driver modules or test harnesses.

#43) Usability Testing: Under Usability Testing, User-friendliness check is done. The application flow is tested to know if a new user can understand the application easily or not, Proper help documented if a user gets stuck at any point. Basically, system navigation is checked in this testing.

#44) Vulnerability Testing: The testing which involves identifying weakness in the software, hardware and the network is known as Vulnerability Testing. Malicious programs, the hacker can take control of the system, if it is vulnerable to such kind of attacks, viruses, and worms.

So it is necessary to check if those systems undergo Vulnerability Testing before production. It may identify critical defects, flaws in the security.

#45) Volume Testing: Volume Testing is a type of Non-Functional Testing performed by the Performance Testing team. The software or application undergoes a huge amount of data and Volume Testing checks the system behavior and response time of the application when the system came across such a high volume of data. This high volume of data may impact the system’s performance and speed of the processing time.

#46) White Box Testing: White Box Testing is based on the knowledge about the internal logic of an application's code. It is also known as Glass box Testing. Internal software and code working should be known for performing this type of testing. Under these tests are based on the coverage of code statements, branches, paths, conditions, etc.

The above-mentioned Software Testing Types are just a part of all different testing types. There are more than 100+ types of testing, but all testing types are not used in all types of projects.

6. AUTOMATION TESTING TUTORIAL: WHAT IS, PROCESS, BENEFITS & TOOLS;

SOFTWARE TEST AUTOMATION OVERVIEW

Automated software testing is carried out with the help of automation testing tools. Automation testing tool is a software application itself with the help of which a tester can write testing scripts and then use this software to test the actual system under testing. Automation tools are used to automate certain sections of manual testing. The biggest advantage of automation testing is that it saves a lot of time and software application can be thoroughly tested in a short span of time. Automation testing tools are available to perform regression, load and stress testing. However, automation testing cannot completely replace manual testing. Beside time, automation testing saves money and effort and helps in improving the accuracy of software.

All aspects of software cannot be covered via automation testing. However, automation testing is of great help in testing GUI aspects, database connectivity etc. It is best to go for automation testing when you are working on large and complex projects where there is a necessity to test certain areas frequently. Automation testing is also good for testing those applications which will be eventually used by several concurrent users. This form of testing helps in saving a lot of time so the team gets more time to focus on the quality of the software.

In order to perform automation testing, you need to know the following details:

1. Which sections of the software require automation testing
2. Which tools will be ideal for this purpose?
3. The process of writing test scripts and executing them
4. You should be able to identify potential bug and know how to create result reports.

It is good to include automation testing to improve the quality of the software but automation testing is not the only way of achieving quality and in no way, it should be considered a replacement for inspections and walkthrough procedures. Many organizations still consider that automation testing is the last way of detecting defects that may have gone unnoticed.

SOFTWARE TEST AUTOMATION STRATEGY

There is no doubt that automation testing makes life a lot easier but just having the right tools is not sufficient to achieve success in automation testing. It is very important to

develop a strategy for automation testing that would clearly define which sections of the software require testing, how this task would be carried out, how and where the scripts will be maintained, how the project would benefit from this activity and how much money would be saved in this process. The more specific you are in defining the strategy; the more successful you will be in achieving your testing goals.

While defining the strategy it is important to break down your aim into smaller goals and check if you are able to achieve what you require with the help of an automation test tool. If the project is too big and you try to test complex areas in the first go then there are chances of making mistakes. So, it is wise to start small initially and then gradually grow.

Automation testing can be carried out in unit testing, integration and system level testing. It is always better to uncover maximum defects in early stages of software development hence it is better to plan automation testing as early as it is possible.

SOFTWARE TEST AUTOMATION AND IT'S RETURN OF INVESTMENT (ROI)

ROI for Software Test Automation is calculated to know how the company would benefit by incorporating this methodology in its testing processes. It gives a fair idea about the cost saving, scope of improvement in efficiency and software quality.

ROI = (Gain- invested amount)/invested amount

The most commonly used methods for calculation of ROI for automation testing are:

- Simple ROI calculation to know how the company would benefit from cost saving. This calculation is of great importance when a company wants to know how it would benefit on monetary basis by incorporating automation testing. The cost of investment would include the amount spent on acquiring the license, hardware, and training for staff, development of scripts, their maintenance and analysis.

- Efficiency ROI calculation tells about the benefits in terms of increased efficiency. Unlike simple ROI, this calculation only looks at time investment gains. When a company already has the automation, testing tools and has been using it for quite some time, there is no need for it to know about Simple ROI calculations because it is not going to make any fresh monetary investment in this case.

- Risk Reduction ROI calculation indicates reduction in risk and scope of improvement of quality. Automation testing saves time and provides team with more time to carry out analysis and carry out ad hoc and exploratory testing this leads to thorough coverage thereby reducing the chances of failure.

TEST CASES TO AUTOMATE

It is difficult and not wise to automate all testing. So, one must clearly determine which test cases should be automated. You must first look at those sections of the software that require repeated testing. Test cases that have to be executed repeatedly and require a lot of

data for execution should be automated. Besides, repetitive tests, you should also focus on automating:

1. Test cases that require multiple data sets for execution and are difficult to conduct manually.

2. Load and stress tests that are difficult to conduct manually.

3. Tests that are executed to check the performance of software on multiple platforms.

4. GUI testing.

TEST CASES NOT TO AUTOMATE

There are certain test cases that are not preferred to be automated. Such as:

1. Test cases that would be executed only once or quite infrequently.

2. Ad hoc or exploratory testing cannot be conducted with the help of automation testing.

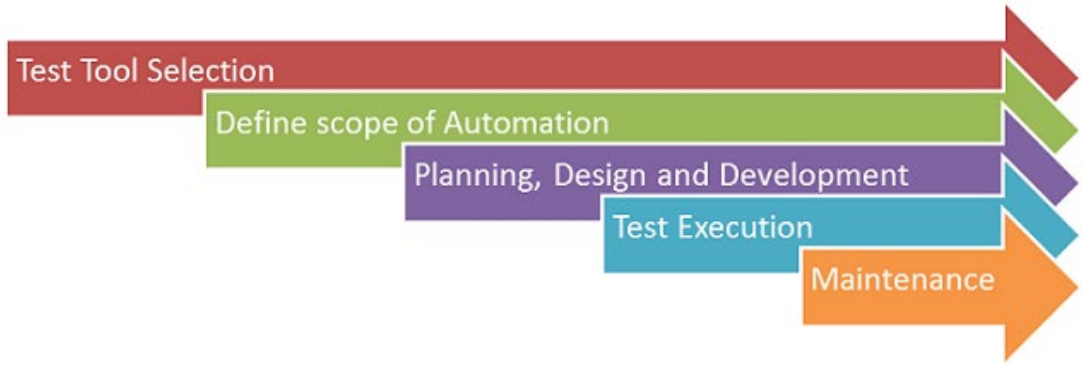
3. Test cases that require only manual execution or a human opinion.

AUTOMATED TESTING PROCESS/ HOW DO WE AUTOMATE?

Automation of a test case begins with defining what you want to test. Always go for those test cases that are independent and stable. It is always better to execute smaller independent test cases rather than few big and complex ones. Large scripts are difficult to maintain and can be difficult to execute if any major changes have been made to the software. If a test script requires a change then it is always easier to make changes to a shorter script than a longer one. Before automating a test case it is better to execute the test case manually. Since automated test cases are those that one needs to use for repetitive testing therefore before designing such a test case it is better to execute it manually and note down all the steps that you would like to record or put down in your test script. This will help you design all the conditions required for through testing of a function. Ensure that you have taken care of all validations in your test scripts. If you are planning to test database transactions, then you need to make sure that database has the necessary data required for testing.

Before executing the test case it is important to understand how the automated software tool has been designed. This is important because once you start testing you don’t want to get interrupted by unknown issues.

Following steps are followed in an Automation Process



Test tool selection: Test Tool selection largely depends on the technology the Application Under Test is built on. For instance, QTP does not support Informatica. So QTP cannot be used for testing Informatica applications. It's a good idea to conduct a Proof of Concept of Tool on AUT.

Define the scope of Automation: The scope of automation is the area of your Application Under Test which will be automated.

Following points help determine scope:

- The features that are important for the business
- Scenarios which have a large amount of data
- Common functionalities across applications
- Technical feasibility
- The extent to which business components are reused
- The complexity of test cases
- Ability to use the same test cases for cross-browser testing

Planning, Design, and Development

During this phase, you create an Automation strategy & plan, which contains the following details-

- Automation tools selected
- Framework design and its features
- In-Scope and Out-of-scope items of automation
- Automation testbed preparation
- Schedule and Timeline of scripting and execution
- Deliverables of Automation Testing

Test Execution: Automation Scripts are executed during this phase. The scripts need input test data before there are set to run. Once executed they provide detailed test reports. Execution can be performed using the automation tool directly or through the Test Management tool which will invoke the automation tool.

EXAMPLE:

Quality center is the Test Management tool which in turn will invoke QTP for execution of automation scripts. Scripts can be executed in a single machine or a group of machines. The execution can be done during the night, to save time.

Maintenance: As new functionalities are added to the System Under Test with successive cycles, Automation Scripts need to be added, reviewed and maintained for each release cycle. Maintenance becomes necessary to improve the effectiveness of Automation Scripts.

FRAMEWORK FOR AUTOMATION

A framework is set of automation guidelines which help in:

Maintaining consistency of Testing

Improves test structuring

Minimum usage of code

Less Maintenance of code

Improve re-usability

Non Technical testers can be involved in code

The training period of using the tool can be reduced

Involves Data wherever appropriate

There are four types of frameworks used in automation software testing:

Data Driven Automation Framework

Keyword Driven Automation Framework

Automation Tool Best Practices

Modular Automation Framework

Hybrid Automation Framework

To get maximum ROI of automation, observe the following: The scope of Automation needs to be determined in detail before the start of the project. This sets expectations from Automation right; Select the right automation tool: A tool must not be selected based on its popularity, but it's fit to the automation requirements; Choose an appropriate framework

Scripting Standards: Standards have to be followed while writing the scripts for Automation. Some of them are:

- Create uniform scripts, comments, and indentation of the code
- Adequate Exception handling - How error is handled on system failure or unexpected behavior of the application.
- User-defined messages should be coded or standardized for Error Logging for testers to understand.

- Measure metrics- Success of automation cannot be determined by comparing the manual effort with the automation effort but by also capturing the following metrics.
- Percent of defects found
- The time required for automation testing for each and every release cycle
- Minimal Time is taken for release
- Customer Satisfaction Index
- Productivity improvement
- The above guidelines if observed can greatly help in making your automation successful.

Following are benefits of automated testing:

- 70% faster than the manual testing
- Wider test coverage of application features
- Reliable in results
- Ensure Consistency
- Saves Time and Cost
- Improves accuracy
- Human Intervention is not required while execution
- Increases Efficiency
- Better speed in executing tests
- Re-usable test scripts
- Test Frequently and thoroughly
- More cycle of execution can be achieved through automation
- Early time to market

HOW TO CHOOSE AN AUTOMATION TOOL?

Automation tools have a very positive impact on the efficiency and productivity of software. It is important to use software test automation tools in order to simplify testing. Certain manual testing processes can be very time consuming. With the help of automation tools you can accomplish more in less time. However, never depend completely on automation tools. You must clearly define what needs to be tested manually and where there is a need to implement automation testing tools. Most well reputed automation testing tools come with their prices, but it is worth deploying them in projects as they are of great help.

Selecting the right tool can be a tricky task. Following criterion will help you select the best tool for your requirement-

- Environment Support
- Ease of use
- Testing of Database
- Object identification
- Image Testing
- Error Recovery Testing
- Object Mapping
- Scripting Language Used
- Support for various types of test - including functional, test management, mobile, etc...
- Support for multiple testing frameworks
- Easy to debug the automation software scripts
- Ability to recognize objects in any environment

- Extensive test reports and results
- Minimize training cost of selected tools

Tool selection is one of biggest challenges to be tackled before going for automation. First, Identify the requirements, explore various tools and its capabilities, set the expectation from the tool and go for a Proof Of Concept.

AUTOMATION TESTING TOOLS

There are tons of Functional and Regression Testing Tools available in the market. Here are some of the best tools currently available:

Ranorex Studio: Ranorex Studio is an all-in-one tool for automating functional UI tests, regression tests, data-driven tests and much more. Ranorex Studio includes an easy to use click-and-go interface to automate tests for web, desktop, and mobile applications.

> Features: Functional UI and end-to-end testing on desktop, web, and mobile; Cross-browser testing; SAP, ERP, Delphi and legacy applications. ; iOS and Android; Run tests locally or remotely, in parallel or distribute on a Selenium Grid; Robust reporting

Mabl: mabl delivers scriptless end-to-end test automation, integrated with your delivery pipeline, so you can focus on improving your app.

> Features: Proprietary machine learning models automatically identify and surface application issues; Tests are automatically repaired when UI changes; Automated regression insights on every build

Selenium: It is a software testing tool used for Regression Testing. It is an open source testing tool that provides playback and recording facility for Regression Testing. The Selenium IDE only supports Mozilla Firefox web browser. It provides the provision to export recorded script in other languages like Java, Ruby, RSpec, Python, C#, etc. It can be used with frameworks like JUnit and TestNG

> It can execute multiple tests at a time; Autocomplete for Selenium commands that are common; Walkthrough tests; Identifies the element using id, name, X-path, etc.; Store tests as Ruby Script, HTML, and any other format; It provides an option to assert the title for every page; It supports selenium user-extensions.js file; It allows to insert comments in the middle of the script for better understanding and debugging

QTP (MicroFocus UFT): QTP is widely used for functional and regression testing, it addresses every major software application and environment. To simplify test creation and maintenance, it uses the concept of keyword driven testing. It allows the tester to build test

cases directly from the application. It is easier to use for a non-technical person to adapt to and create working test cases. It fix defects faster by thoroughly documenting and replicating defects for developer.

Collapse test creation and test documentation at a single site; Parameterization is easy than WinRunner; QTP supports .NET development environment; It has better object identification mechanism; It can enhance existing QTP scripts without "Application Under Test" is available, by using the active screen;

Rational Functional Tester: It is an Object-Oriented automated Functional Testing tool that is capable of performing automated functional, regression, data-driven testing and GUI testing. The main features of this tool are: It supports a wide range of protocols and applications like Java, HTML, NET, Windows, SAP, Visual Basic, etc.; It can record and replay the actions on demand; It integrates well with source control management tools such as Rational Clear Case and Rational Team Concert integration; It allows developers to create keyword associated script so that it can be re-used; Eclipse Java Developer Toolkit editor facilitates the team to code test scripts in Java with Eclipse; It supports custom controls through proxy SDK (Java/.Net); It supports version control to enable parallel development of test scripts and concurrent usage by geographically distributed team

WATIR: It is an open source testing software for regression testing. It enables you to write tests that are easy to read and maintain. Watir supports only internet explorer on windows while Watir webdriver supports Chrome, Firefox, IE, Opera, etc.

It supports multiple browsers on different platforms; Rather than using proprietary vendor script, it uses a fully-featured modern scripting language Ruby; It supports your web app regardless of what it is developed in

SilkTest: Silk Test is designed for doing functional and regression testing. For e-business application, silk test is the leading functional testing product. It is a product of Segue Software takeover by Borland in 2006. It is an object-oriented language just like C++. It uses the concept of an object, classes, and inheritance. Its main feature includes: It consists of all the source script files; It converts the script commands into GUI commands. On the same machine, commands can be run on a remote or host machine; To identify the movement of the mouse along with keystrokes, Silktest can be executed. It can avail both playback and record method or descriptive programming methods to get the dialogs; It identifies all controls and windows of the application under test as objects and determines all of the attributes and properties of each window

7. AUTOMATION TESTING VS. MANUAL TESTING: WHAT'S THE DIFFERENCE?

Both manual and automation testing have their own share of benefits and limitations. Below is a comparison between these two types of testing.

Manual Testing	Automation Testing
Manual testing is a type of testing in which test case execution is performed manually by humans.	Automation testing is a type of testing in which automated test case execution is performed using different automation tools.
Manual test case execution is very tedious and time consuming.	Automated test case execution is very fast, only the initial test framework and test script creation takes time.
It is more suited for user-interface, adhoc and exploratory testing.	It is not suited for user-interface, adhoc and exploratory testing.
Manual testing is not suitable for performing load testing and testing which requires frequent test executions.	Load testing can be done using automation testing. Also, it is suited for regression tests and test which require frequent test case execution.
The manual test cases are required to be run sequentially.	Automated test scripts can be run in parallel as well using distributed testing across different machines.
It is considered less reliable because of human error.	Following right approach and standards for test script creation leads to reliable automated testing.

WHAT ARE THE DIFFERENT TEST DESIGN TECHNIQUES?

Test design techniques are standards of test designing that allow the creation of systematic and widely accepted test cases. These techniques are based on the different scientific models and over the years experiences of many QA professionals.
The test design techniques can be broadly categorized into two parts – “Static test design technique” and “Dynamic test design technique”.

STATIC TEST DESIGN TECHNIQUES

The Static test design techniques are the testing techniques that involve testing without executing the code or the software application. So, basically static testing deals with Quality Assurance, involving reviewing and auditing of code and other design documents. The

various static test design techniques can be further divided into two parts – “Static testing performed manually” and “Static testing using tools”.

1.1. MANUAL STATIC DESIGN TECHNIQUES

1.1.1. Walk through – A Walk-through is step by step presentation of different requirement and design documents by their authors with the intent of finding defects or any missing pieces in the documents.

1.1.2 Informal reviews – As the name suggests, an informal review done by an individual without any process or documentation.

1.1.3 Technical reviews – A technical review involves reviewing the technical approach used during the development process. It is more of a peer review activity and less formal as compared to audit and inspection.

1.1.4 Audit – An audit is a formal evaluation of the compliance of the different processes and artifacts with standards and regulations. It is generally performed by an external or independent team or person.

1.1.5 Inspection – An inspection is a formal and documented process of reviewing the different documents by experts or trained professionals.

1.1.6 Management review – It is a review performed on the different management documents like project management plans, test plans, risk management plans etc.

1.2. STATIC DESIGN TECHNIQUES USING TOOLS (AUTOMATED)

2.1 Static analysis of code – The static analysis techniques for source code evaluation using tools are:

a) Control flow analysis – The control flow analysis requires analysis of all possible control flows or paths in the code.
b) Data flow analysis – The data flow analysis requires the analysis of data in the application and its different states.
c) Compliance to coding standard – This evaluates the compliance of the code with the different coding standards.
d) Analysis of code metrics – The tool used for static analysis is required to evaluate the different metrics like lines of code, complexity, code coverage etc.

DYNAMIC TEST DESIGN TECHNIQUES

Dynamic test design techniques involve testing by running the system under test. In this technique, the tester provides input data to the application and executes it, in order to verify its different functional and non-functional requirements.

a) Specification-based – Specification-based test design techniques are also referred to as black box testing. These involve testing based on the specification of the system under test without knowing its internal architecture. The different types of specification-based test design or black box testing techniques are – “Equivalence partitioning”, “Boundary value analysis”, “Decision tables”, “Cause-effect graph”, “State transition testing” and “Use case testing”.
b) Structure based – Structure-based test design techniques are also referred to as white box testing. In these techniques, the knowledge of code or internal architecture of the system is required to carry out the testing. The various kinds of testing structure-based or white testing techniques are – “Statement testing”, “Decision testing/branch testing”, “Condition testing”, “Multiple condition testing”, “Condition determination testing” and “Path testing”.
c) Experienced based – The experienced based techniques as the name suggest does not require any systematic and exhaustive testing. These are completely based on the experience or intuition of the tester. The two most common forms of experienced-based testing are – Adhoc testing and exploratory testing.

8. WHAT IS REGRESSION TESTING? DEFINITION, TOOLS, METHOD, AND EXAMPLE

Regression Testing is a type of testing that is done to verify that a code change in the software does not impact the existing functionality of the product. This is to make sure the product works fine with new functionality, bug fixes or any change in the existing feature. Previously executed test cases are re-executed in order to verify the impact of change.

Regression Testing is a Software Testing type in which test cases are re-executed in order to check whether the previous functionality of the application is working fine and the new changes have not introduced any new bugs.

This test can be performed on a new build when there is a significant change in the original functionality that too even in a single bug fix.

Regression means retesting the unchanged parts of the application.

REGRESSION TEST OVERVIEW

Regression test is like a verification method. Test cases are generally automated as test cases are required to execute again and again and running the same test cases again and again manually is time-consuming and tedious one too.

Exercise:

Consider a product X, in which one of the functionality is to trigger confirmation, acceptance, and dispatched emails when Confirm, Accept and Dispatch buttons are clicked.

Some issue occurs in the confirmation email and in order to fix the same, some code changes are done. In this case, not only the Confirmation emails need to be tested but Acceptance and Dispatched emails also needs to be tested to ensure that the change in the code has not affected them.

Regression Testing is not dependent on any programming language like Java, C++, C#, etc. It is a testing method which is used to test the product for modifications or for any updates being done. It verifies that any modification in a product does not affect the existing modules of the product. Verifying that the bugs are fixed and the newly added features have not created any problem in the previous working version of the software.

Testers perform Functional Testing when a new build is available for verification. The intent of this test is to verify the changes made in the existing functionality and the newly added functionality as well. When this test is done, the tester should verify whether the existing functionality is working as expected and the new changes have not introduced any defect in functionality that was working before this change.

Regression test should be a part of the Release Cycle and must be considered in the test estimation.

WHEN TO PERFORM THIS TEST?

Regression Testing is usually performed after verification of changes or new functionality. But this is not the case always. For the release that is taking months to complete, regression tests must be incorporated in the daily test cycle. For weekly releases, regression tests can be performed when the Functional Testing is over for the changes. Regression checking is a variation of retest (which is simply to repeat a test). When Retesting, the reason can be anything. Say, you were testing a particular feature and it was the end of the day- you could not finish testing and had to stop the process without deciding if the test passed/failed. The next day when you come back, you perform the test once more – that means you are repeating a test you performed before. The simple act of repeating a test is a Retest.

Regression test at its core is a retest of sorts. It is only for the special occasion that something in the application/code has changed. It might be code, design or anything at all that dictates the overall framework of the system.

A Retest that is conducted in this situation to make sure that the said change has not made an impact on anything that was already working before is called Regression Test. The most common reasons why this might be conducted are because new versions of the code have been created (increase in scope/requirement) or bugs have been fixed.

To perform the test execution do we need a tool? How is Regression Testing performed? Can this Testing be performed manually?

To begin with, Test execution is a simple act of using your Test cases and performing those steps on the AUT, supplying the test data and comparing the result obtained on the AUT with the expected result mentioned in your test cases. Depending on the comparison result, we set the status of the test case pass/fail. Test execution is as simple as that, there are no special tools necessary for this process.

Automated Regression Test is the testing area where we can automate most of the testing efforts. We run all the previously executed test cases on a new build. This means that we have a test case set available and running these test cases manually is time-consuming. We know the expected results, so automating these test cases is time-saving and is an efficient regression test method. The extent of automation depends upon the number of test cases that are going to remain applicable overtime.

If test cases are varying from time to time, the application scope goes on increasing and then automation of regression procedure will be a waste of time.

Most of the Regression test tools are record and playback type. You will record the test cases by navigating through the AUT (application under test) and verify whether the expected results are coming or not.

Recommended Tool

#1) Ranorex Studio: Enhance Software Quality and maximize your resources with this powerful Automated Regression Testing tool. You can execute more test cases in a fraction of the time with Ranorex which is up to 78% efficiency increase over Manual Testing.

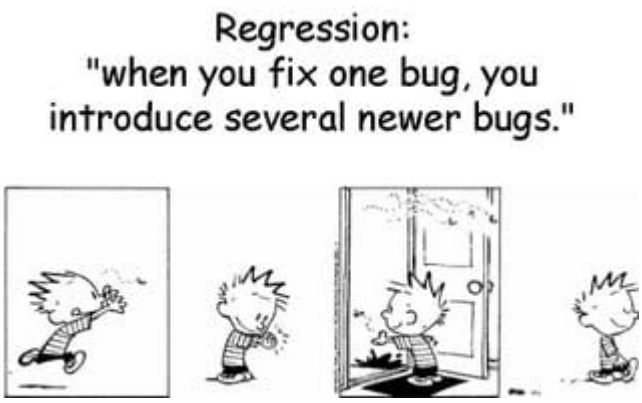
Other Tools

- Selenium
- Katalon Studio
- AdventNet QEngine
- Regression Tester
- vTest
- Watir
- actiWate
- Rational Functional Tester
- SilkTest
- TimeShiftX

Most of these are Functional and Regression test tools.

Adding and updating Regression test cases in an Automation test suite is a cumbersome task. While selecting an Automation tool for Regression tests, you should check if the tool allows you to add or update the test cases easily. In most cases, we need to update automated Regression test cases frequently due to frequent changes in the system.

WHY THE REGRESSION TEST?



Regression is initiated when a programmer fixes any bug or adds a new code for new functionality to the system. There can be many dependencies in the newly added and existing functionality. It is a quality measure to check whether the new code complies with the old code so that the unmodified code is not getting affected. Most of the time the testing team has the task to check the last-minute changes in the system.

In such a situation, testing only affected application area is necessary to complete the testing process on time by covering all the major system aspects.

This test is very important when there is a continuous change/improvement added in the application. The new functionality should not negatively affect the existing tested code. Regression is required to find the bugs that occurred because of a change in the code. If this testing is not done, the product might get critical issues in the live environment and that indeed can lead the customer into trouble.

While testing any online website, a tester reports an issue that the Price of the Product is not shown correctly i.e. it shows a lesser price than the actual price of the Product, and it needs to be fixed soon. Once the developer fixes the issue, it needs to be re-tested and Regression Testing is also required as verifying the price at the reported page would have got corrected but it might be showing an incorrect price at the summary page where the total is shown along with the other charges or the mail sent to the customer still has the incorrect price. Now, in this case, the customer will have to bear the loss if this testing is not performed as the site calculates the total cost with the incorrect price and the same price goes to a customer by email. Once the customer accepts, the Product is sold online at a lower price, it will be a loss for the customer.

So, this testing plays a big role and is very much required and important as well.

TYPES OF REGRESSION TESTING:

Unit Regression Partial Regression Complete Regression

#1) Unit Regression: Unit Regression is done during the Unit Testing phase and code is tested in isolation i.e. any dependencies on the unit to be tested are blocked so that the unit can be tested individually without any discrepancy.

#2) Partial Regression: Partial Regression is done to verify that the code works fine even when the changes have been done in the code and that unit is integrated with the unchanged or already existing code.

#3) Complete Regression: Complete Regression is done when a change in the code is done on a number of modules and also if the change impact of a change in any other module is uncertain. The product as a whole is regressed to check any changes because of the changed code.

HOW MUCH REGRESSION IS REQUIRED?

This depends upon the scope of newly added features. If the scope of a fix or feature is too large, then the application area getting affected is also quite large and the testing should be performed thoroughly including all the application test cases. But this can be effectively decided when the tester gets input from a developer about the scope, nature, and the amount of change. As these are repetitive tests, test cases can be automated so that a set of test cases alone can be easily executed on a new build.

Regression test cases need to be selected very carefully so that maximum functionality is covered in a minimum set of test cases. These set of test cases need continuous improvements for newly added functionality. It becomes very difficult when the application scope is very huge and there are continuous increments or patches to the system. In such cases, selective tests need to be executed in order to save testing cost and time. These selective test cases are picked based on the enhancements done to the system and the parts where it can affect the most.

WHAT DO WE DO IN REGRESSION CHECK?

Re-run the previously conducted tests and compare the current results with previously executed test results. This is a continuous process performed at various stages throughout the software testing lifecycle.

A best practice is to conduct a Regression test after the Sanity or Smoke Testing and at the end of Functional testing for a short release.

In order to conduct effective testing, a regression Test Plan should be created. This plan should outline the regression testing strategy and the exit criteria. Performance Testing is also a part of this test to make sure that the system performance is not affected due to the changes made in the system components.

Best practices: Run automated test cases every day in the evening so that any regression side effects can be fixed in the next day build. This way it reduces the release risk by covering almost all regression defects at an early stage rather than finding and fixing those at the end of the release cycle.

REGRESSION TESTING TECHNIQUES

Retest all Regression Test Selection Test case Prioritization Hybrid

#1) Retest All: As the name itself suggests, the entire test cases in the test suite are re-executed to ensure that there are no bugs that have occurred because of a change in the code. This is an expensive method as it requires more time and resources when compared to the other techniques.

#2) Regression Test Selection: In this method, test cases are selected from the test suite to be re-executed. Not the entire suite is re-executed. The selection of test cases is done on the basis of code change in the module. Test cases are divided into two categories, one is Reusable test cases and another one is Obsolete test cases. The reusable test cases can be used in future regression cycles whereas obsolete ones are not used in the upcoming regression cycles.

#3) Test Case Prioritization: Test cases with high Priority are executed first than the ones with medium and low priority. The priority of test case depends on its criticality and its impact on the product and also on the functionality of the product which is used more often.

#4) Hybrid: The hybrid technique is a combination of Regression test selection and Test case Prioritization. Rather than selecting the entire test suite, select only the test cases which are re-executed depending on their priority.

HOW TO SELECT A REGRESSION TEST SUITE?

Most of the bugs found in the production environment occur because of the changes did or bugs fixed at the eleventh hour i.e. the changes done at a later stage. The bug fix at the last stage might create other issues/bugs in the Product. That's why Regression checking is very important before releasing a Product.

Below is a list of test cases that can be used while performing this Test:

- Functionalities which are frequently used.
- Test cases that cover the module where the changes have been done.
- Complex test cases.
- Integration test cases which include all the major components.
- Test cases for the core functionality or feature of the Product.
- Priority 1 and Priority 2 test cases should be included.
- Test cases that frequently fail or recent testing defects were found in the same.

HOW TO PERFORM REGRESSION TESTING?

Now that we have established what regression means, it is apparent that it is testing also – simply repeating in a specific situation for a specific reason. Therefore, we can safely derive that the same method applies for testing in the first place can be applied to this too.

Therefore, if testing can be done manually then Regression Testing can be too. The use of a tool is not mandatory. However, as time goes on applications get piled on with more and more functionality which keeps increasing the scope of regression. To make the most of the time, this testing is most often Automated.

EXERCISE 1:

Prepare a Test suite for Regression considering the points mentioned above > "How to select Regression Test suite"?

Automate all the test cases of the test suite.

Update the Regression suite whenever it is required like if any new defect which is not covered in the test case is found, and a test case for the same should be updated in the test suite so that the testing is not missed for the same next time. The regression test suite should be managed properly by continuously updating the test cases.

Execute the Regression test cases whenever there is any change in the code, the bug is fixed, new functionality is added, an enhancement to the existing functionality is done, etc.

Create a test execution Report which includes the Pass/Fails status of the executed test cases.

EXERSICE 2:

Please examine the below situation. Try and make observations yourself first and see what conclusions can be drawn from it.

Release 1 Statistics	
Application Name	XYZ
Version/Release Number	1
No. of Requirements (Scope)	10

Release 1 Statistics	
No. of Test Cases/Tests	100
No. of days it takes to Develop	5
No. of days it takes to Test	5
No. of Testers	3
Release 2 Statistics	
Application Name	XYZ
Version/Release Number	2
No. of Requirements (Scope)	10+ 5 new Requirements
No. of Test cases/Tests	100+ 50 new
No. of days it takes to Develop	2.5 (since this half the amount of work than earlier)
No. of days it takes to Test	5(for the existing 100 TCs) + 2.5 (for new Requirements)
No. of Testers	3

Release 3 Statistics	
Application Name	XYZ

Release 3 Statistics	
Version/Release Number	3
No. of Requirements (Scope)	10+ 5 + 5 new requirements
No. of Test cases/Tests	100+ 50+ 50 new
No. of days it takes to Develop	2.5 (since this half the amount of work than earlier)
No. of days it takes to Test	7.5 (for the existing 150 TCs) + 2.5 (for new Requirements)
No. of Testers	3

The following are the observations can be made from the above situation:

- As the releases grow the functionality grows.
- The development time does not necessarily grow with releases, but the testing time does
- No company/its management will be ready to invest more time in testing and less for development

We cannot even reduce the time it takes to test by increasing the test team size because more people means more money and new people also means lots of training and maybe also a compromise in quality as the new people might not be at par with the required knowledge levels immediately.

The other alternative clearly is to reduce the amount of regression. But that could be risky for the software product.

For all these reasons, Regression Testing is a good candidate for Automation Testing, but it does not have to be done only that way.

BASIC STEPS TO PERFORM REGRESSION TESTS

Every time the software undergoes a change and a new version/release comes up, the following are the steps you can take to carry out this type of testing:

- Understand what kind of changes have been made to the software
- Analyze and determine what modules/parts of the software might be impacted – the development and BA teams can be instrumental in providing this information
- Take a look at your test cases and determine if you will have to do a full, partial or unit regression. Identify the ones that will fit your situation
- Schedule the time and test away!

REGRESSION IN AGILE

Agile is an adaptive approach that follows an iterative and incremental method. The product is developed in short iterations called sprint which lasts for 2- 4 weeks. In agile, there is a number of iterations, hence this testing plays a significant role as the new functionality or code change is done in the iterations.

The Regression test suite should be prepared from the initial phase and should be updated with each sprint.

In Agile, Regression check is covered under two categories:

Sprint Level Regression End to End Regression

#1) Sprint Level Regression: Sprint Level Regression is done mainly for the new functionality or the enhancement that is done in the latest sprint. Test cases from the test suite are selected as per the newly added functionality or the enhancement that is done.

#2) End-to-End Regression: End-to-End Regression includes all the test cases that are to be re-executed to test the complete product end to end by covering all the core functionalities of the Product.

As Agile has short sprints and it goes on, it is very much required to automate the test suite, the test cases are executed again and that too needs to be completed in a short span of time. Automating the test cases reduces the time of execution and defect slippage.

Advantages

- It improves the quality of the Product.
- It ensures that any bug fix or enhancement that is done does not impact the existing functionality of the Product.
- Automation tools can be used for this testing.
- It makes sure that issues that are already fixed do not occur again.

Disadvantages

- Though there are several advantages, there are some disadvantages as well. They are:

- It has to be done for a small change in the code as well because even a small change in the code can create issues in the existing functionality.
- If in case automation is not used in the Project for this testing, it will be a time consuming and tedious task to execute the test cases again and again.

REGRESSION OF GUI APPLICATION

It is difficult to perform a GUI (Graphical User Interface) Regression test when the GUI structure is modified. The test cases written on old GUI either become obsolete or need to be modified.

Re-using the regression test cases means GUI test cases are modified according to the new GUI. But this task becomes a cumbersome one if you have a large set of GUI test cases.

Difference Between Regression And Re-testing

Re-testing is done for the test cases which fail during the execution and the bug raised for the same has been fixed whereas Regression check is not limited to the bug fix as it covers other test cases as well to ensure that the bug fix has not impacted any other functionality of the Product.

REGRESSION TEST PLAN TEMPLATE (TOC)

Let's take a look at each of them in detail.

#1) DOCUMENT HISTORY:

Document history consists of a record of the first draft and all the updated ones in the below-given format.

Version	Date	Author	Comment
1	DD/MM/YY	ABC	Approved
2	DD/MM/YY	ABC	Updated for the added feature

#2) REFERENCES:

References column keep a track of all the reference documents used or required for the Project while creating a test plan.

No	Document	Location
1	SRS document	Shared drive

#3) REGRESSION TEST PLAN:

3.1. Introduction: This document describes the change/update/enhancement in the Product to be tested and the approach used for this testing. All the code changes, enhancements, updates, added features are outlined to be tested. Test cases used for Unit Testing and Integration Testing can be used to create a test suite for Regression.

3.2. Purpose: Purpose of Regression Test Plan is to describe what exactly and how testing would be performed to accomplish the results. Regression check is done to ensure that no other functionality of the product is hampered because of the code change.

3.3. Test Strategy: Test Strategy describes the approach which will be used to perform this testing and that includes the technique that will be used, what will be the completion criteria, who will be performing which activity, who will write the test scripts, which regression tool will be used, steps to cover the risks like resource crunch, delay in production, etc.

3.4. Features to be tested: Feature/components of the product to be tested are listed here. In regression, all the test cases are re-executed or the ones which affect the existing functionality are chosen depending on the fix/update or enhancement done.

3.5. Resource Requirement:

3.5.1. Hardware Requirement: Hardware Requirement is identified here like computers, laptop, Modems, Mac book, Smartphone, etc.

3.5.2. Software Requirement: Software Requirement is identified like which Operating system and browsers will be required.

3.6. Test Schedule: Test schedule defines the estimated time for performing the testing activities.

EXERCISE:

How many resources will perform a testing activity and that too in how much time?

3.7. Change Request: CR details are mentioned for which Regression would be performed.

S.No	CR Description	Regression Test Suite
1		
2		

3.8. Entry/Exit Criteria

3.8.1. Entry Criteria for this testing: Entry criteria for the Product to start Regression check are defined.

EXAMPLE:

Coding changes/enhancement/addition of new feature should be completed.

Regression test Plan should be approved.

3.8.2. Exit Criteria for this testing: Here the exit criteria for Regression are defined.

EXAMPLE:

Regression testing should be completed. Any new critical bugs found during this testing should be closed.

Test Report should be ready.

3.9. Test Cases: Regression Test cases are defined here.

3.10. Risk/Assumptions: Any risk & assumptions are identified and a contingency plan is prepared for the same.

3.11. Tools: Tools to be used in the Project are identified. Such as:

Automation tool Bug Reporting tool

#4) Approval/Acceptance: Names and Designation of the people are listed here:

Name	Approved/Rejected	Signature	Date

A lot of automation tools are available for automating the regression test cases, however, a tool should be selected as per the Project requirement. A tool should have the ability to update the test suite as the Regression test suite needs to be updated frequently.

WHAT IS THE DIFFERENCE BETWEEN REGRESSION AND RETESTING

Let's see the difference between Regression and Retesting. This might be one of the top 5 interview questions for freshers. Most of the testers have confusion with Regression and Retesting. Here in this post, we will show case the difference between regression and retesting with practical example to understand clearly.

REGRESSION TESTING: Repeated testing of an already tested program, after modification, to discover any defects introduced or uncovered as a result of the changes in the software being tested or in another related or unrelated software components.

- Usually, we do regression testing in the following cases:
- New functionalities are added to the application
- Change Requirement (In organizations, we call it as CR)
- Defect Fixing

- Performance Issue Fix
- Environment change (E.g.. Updating the DB from MySQL to Oracle)

RETESTING: To ensure that the defects which were found and posted in the earlier build were fixed or not in the current build.

Say, Build 1.0 was released. Test team found some defects (Defect Id 1.0.1, 1.0.2) and posted.

Build 1.1 was released, now testing the defects 1.0.1 and 1.0.2 in this build is retesting.

EXERCISE 1:

Tto showcase the difference between Regression and Retesting, let's take two scenarios.

Case 1: Login Page – Login button not working (Bug)

Case 2: Login Page – Added "Stay signed in" checkbox (New feature)

In Case 1, Login button is not working, so tester reports a bug. Once the bug is fixed, testers test it to make sure whether the Login button is working as per the expected result.

In Case 2, tester tests the new feature to ensure whether the new feature (Stay signed in) is working as intended.

Case 1 comes under Re-testing. Here tester retests the bug which was found in the earlier build by using the steps to reproduce which were mentioned in the bug report.

Also in the Case 1, tester tests other functionalities which are related to login button which we call as Regression Testing.

Case 2 comes under Regression Testing. Here tester tests the new feature (Stay signed in) and also tests the relevant functionalities. Testing the relevant functionalities while testing the new features come under Regression Testing.

EXAMPLE 1:

Imagine, An Application Under Test has three modules namely Admin, Purchase and Finance. Finance module depends on Purchase module. If a tester found a bug on Purchase module and posted. Once the bug is fixed, the tester needs to do Retesting to verify whether the bug related to the Purchase is fixed or not and also tester needs to do Regression Testing to test the Finance module which depends on the Purchase module.

SOME OTHER DIFFERENCES BETWEEN REGRESSION AND RETESTING:

Retesting done on failed test cases whereas Regression Testing done on passed test cases.

Retesting makes sure that the original defect has been corrected whereas Regression Testing makes sure that there are no unexpected side effects.

9. WHAT IS A TEST SCENARIO?

A Test Scenario is defined as any functionality that can be tested. It is a collective set of test cases which helps the testing team to determine the positive and negative characteristics of the project.

Test Scenario gives a high-level idea of what we need to test.

EXERCISE 1:

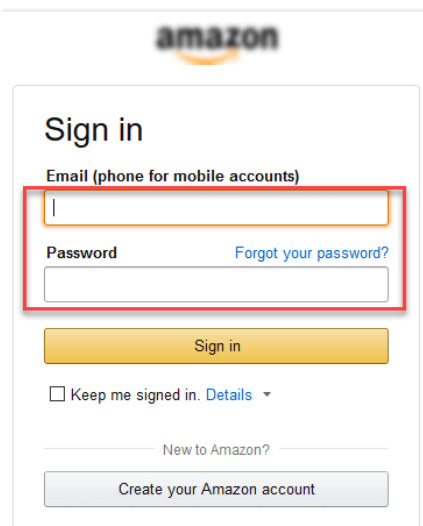
Test Scenario

For an eCommerce Application, a few test scenarios would be

Test Scenario 1: Check the Search Functionality

Test Scenario 2: Check the Payments Functionality

Test Scenario 3: Check the Login Functionality



WHY DO WE WRITE TEST SCENARIO?

The main reason to write a test scenario is to verify the complete functionality of the software application. It also helps you to ensure that the business processes and flows are as per the functional requirements.

Test Scenarios can be approved by various stakeholders like Business Analyst, Developers, Customers to ensure the Application Under Test is thoroughly tested. It ensures that the software is working for the most common use cases.

They serve as a quick tool to determine the testing work effort and accordingly create a proposal for the client or organize the workforce. They help determine the most critical end-to-end transactions or the real use of the software applications.

Once these Test Scenarios are finalized, test cases can be easily derived from the Test Scenarios.

EXAMPLE 1: BEST PRACTICES OF CREATING A TEST SCENARIO

	A	B	C
1			
2			**Test Scenario Template**
3			
4	* Test Cases field is Optional		
5			
6	Test Scenario #	Requiremen	Test Scenario Description
7	1	S1.1	Check the Login Functionality
8	2	S1.2	Check the Search Functionality
9	3	S1.3	Check the Product Description Page
10	4	S1.4	Check the Payments Functionality
11	5	S1.5	Check the Order History

Test scenarios are mostly single line statement that tells what should be tested. Scenario description should be simple and easy to understand. A careful assessment of the stated requirements should be done. The required tools and resources for testing need to be accumulated before the beginning of the testing process.

WHAT IS A TEST CASE?

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.

EXAMPLES:

Test cases for the Test Scenario: "Check the Login Functionality" would be

Check system behavior when valid email id and password is entered.

Check system behavior when invalid email id and valid password is entered.

Check system behavior when valid email id and invalid password is entered.

Check system behavior when invalid email id and invalid password is entered.

Check system behavior when email id and password are left blank and Sign in entered.

Check Forgot your password is working as expected

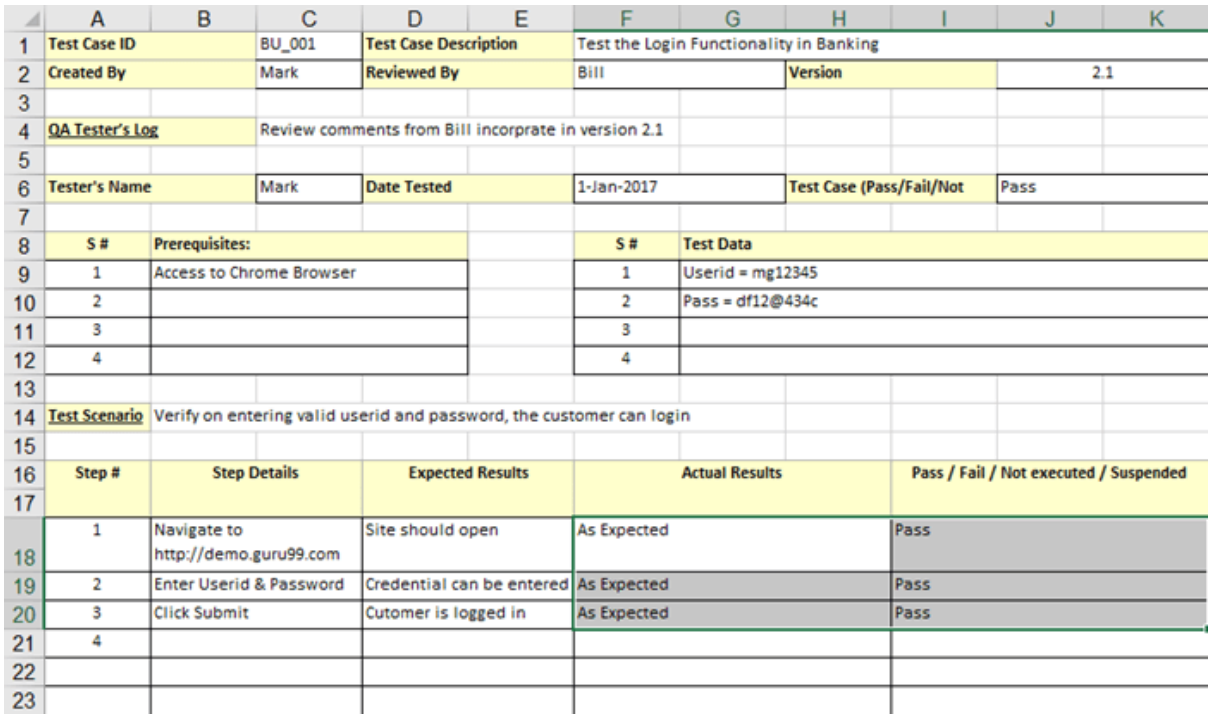
Check system behavior when valid/invalid phone number and password is entered.

Check system behavior when "Keep me signed" is checked

WHY DO WE WRITE TEST CASES?

Test cases help to verify conformance to applicable standards, guidelines and customer requirements. Helps you to validate expectations and customer requirements. Increased control, logic, and data flow coverage. You can simulate 'real' end user scenarios. Exposes errors or defects. When test cases are written for test execution, the test engineer's work will be organized better and simplified.

EXAMPLE 1: BEST PRACTICES OF CREATING TEST CASES



	A	B	C	D	E	F	G	H	I	J	K
1	Test Case ID		BU_001	Test Case Description		Test the Login Functionality in Banking					
2	Created By		Mark	Reviewed By		Bill		Version		2.1	
3											
4	QA Tester's Log		Review comments from Bill incorprate in version 2.1								
5											
6	Tester's Name		Mark	Date Tested		1-Jan-2017		Test Case (Pass/Fail/Not		Pass	
7											
8	S #	Prerequisites:				S #	Test Data				
9	1	Access to Chrome Browser				1	Userid = mg12345				
10	2					2	Pass = df12@434c				
11	3					3					
12	4					4					
13											
14	Test Scenario	Verify on entering valid userid and password, the customer can login									
15											
16	Step #	Step Details		Expected Results		Actual Results			Pass / Fail / Not executed / Suspended		
17											
18	1	Navigate to http://demo.guru99.com		Site should open		As Expected			Pass		
19	2	Enter Userid & Password		Credential can be entered		As Expected			Pass		
20	3	Click Submit		Cutomer is logged in		As Expected			Pass		
21	4										
22											
23											

Test Cases should be transparent and straightforward; Create Test Case by keeping the end user in the mind; Avoid test case repetition; You need to make sure that you will write test cases to check all software requirements mentioned in the specification document; Never assume functionality and features of your software application while preparing a test case; Test Cases must be readily identifiable

Here, are significant differences between Test scenario and a Test Case

Test Scenario	Test Case
A test scenario contains high-level documentation which describes an end to end functionality to be tested.	Test cases contain definite test steps, data, expected results for testing all the features of an application.
It focuses on more "what to test" than "how to test".	A complete emphasis on "what to test" and "how to test.".
Test scenarios are a one-liner. So, there is always the possibility of ambiguity during the testing.	Test cases have defined a step, pre-requisites, expected result, etc. Therefore, there is no ambiguity in this process.
Test scenarios are derived from test artifacts	Test case is mostly derived from test

like BRS, SRS, etc.	scenarios. Multiple Test case can be derived from a single Test Scenario
It helps in an agile way of testing the end to end functionality	It helps in exhaustive testing of an application
Test scenarios are high-level actions.	Test cases are low-level actions.
Comparatively less time and resources are required for creating & testing using scenarios.	More resources are needed for documentation and execution of test cases.

KEY DIFFERENCE

- Test Case is a set of actions executed to verify particular features or functionality whereas Test Scenario is any functionality that can be tested.
- Test Case is mostly derived from test scenarios while Test Scenarios are derived from test artifacts like BRS and SRS.
- Test Case helps in exhaustive testing of an application whereas Test Scenario helps in an agile way of testing the end to end functionality.
- Test Cases are focused on what to test and how to test while Test Scenario is more focused on what to test.
- Test Cases are low-level actions whereas Test Scenarios are high-level actions.
- Test Case requires more resources and time for test execution while Test Scenario require fewer resources and time for test execution.
- Test Case includes test steps, data, expected results for testing whereas Test Scenario includes an end to end functionality to be tested.

10. HOW TO WRITE A TEST CASE

Ideally, a test case management tool should be used for managing the test cases and test execution cycles, such as Quality Center (HP QC), JIRA etc. But for smaller projects, many organizations still prefer to use spreadsheets for avoiding the overhead of maintaining and configuring a tool.

Test cases are created for a particular test scenario in order to verify whether the features of an application are working as intended or not. Test cases are the set of positive and negative executable steps of a test scenario which has a set of pre-conditions, test data, expected result, post-conditions and actual results.

Now we will see the different fields of a test case – mandatory as well optional. Assume we need to write test cases for a scenario.

Let's discuss the main fields of a test case (Example case 1):

PROJECT NAME: Name of the project the test cases belong to

MODULE NAME: Name of the module the test cases belong to

REFERENCE DOCUMENT: Mention the path of the reference documents (if any such as Requirement Document, Test Plan, Test Scenarios etc.,)

CREATED BY: Name of the Tester who created the test cases

DATE OF CREATION: When the test cases were created

REVIEWED BY: Name of the Tester who created the test cases

DATE OF REVIEW: When the test cases were reviewed

EXECUTED BY: Name of the Tester who executed the test case

DATE OF EXECUTION: When the test case was executed

TEST CASE ID: Each test case should be represented by a unique ID. It's good practice to follow some naming convention for better understanding and discrimination purpose.

TEST SCENARIO: Test Scenario ID or title of the test scenario.

TestCaseId – This field uniquely identifies a test case. It is mapped with automation scripts (if any) to keep a track of automation status. The same field can be used for mapping with the test scenarios for generating a traceability matrix. E.g. – GoogleSearch_1

PRE-CONDITION: Conditions which needs to meet before executing the test case.

TEST STEPS: Mention all the test steps in detail and in the order how it could be executed.

TEST DATA: The data which could be used an input for the test cases.

EXPECTED RESULT: The result which we expect once the test cases are executed. It might be anything such as Home Page, Relevant screen, Error message etc.,

POST-CONDITION: Conditions which needs to achieve when the test case was successfully executed.

ACTUAL RESULT: The result which system shows once the test case was executed.

STATUS: If the actual and expected results are same, mention it as Passed. Else make it as Failed. If a test fails, it has to go through the bug life cycle to be fixed.

EXERCISE: CASE TEMPLATE 1

Project Name:	**Google Email**
Module Name:	**Login**
Reference Document:	**If any**
Created by:	**Rajkumar**
Date of creation:	**DD-MMM-YY**
Date of review:	**DD-MMM-YY**

TEST CASE ID	TEST SCENARIO	TEST CASE	PRE-CONDITION	TEST STEPS	TEST DATA	
TC_LOGIN_001	Verify the login of Gmail	Enter valid User Name and valid Password	1. Need a valid Gmail Account to do login	1. Enter User Name	<Valid User Name>	
				2. Enter Password	<Valid Password>	Suc
				3. Click "Login" button		
TC_LOGIN_001	Verify the login of Gmail	Enter valid User Name and invalid Password	1. Need a valid Gmail Account to do login	1. Enter User Name	<Valid User Name>	A m
				2. Enter Password	<Invalid Password>	pass
				3. Click "Login" button		mat
TC_LOGIN_001	Verify the login of Gmail	Enter invalid User Name and valid Password	1. Need a valid Gmail Account to do login	1. Enter User Name	<Invalid User Name>	A m
				2. Enter Password	<Valid Password>	pass
				3. Click "Login" button		mat
TC_LOGIN_001	Verify the login of Gmail	Enter invalid User Name and invalid Password	1. Need a valid Gmail Account to do login	1. Enter User Name	<Invalid User Name>	A m
				2. Enter Password	<Invalid Password>	pass
				3. Click "Login" button		mat

Example scenario 1: Verify the login of Gmail account	1. Enter valid User Name and valid Password 2. Enter valid User Name and invalid Password 3. Enter invalid User Name and valid Password 4. Enter invalid User Name and invalid Password

EXERCISE: CASE TEMPLATE 2

Example scenario 2:

Verify Google Search bar module

SCREEN NAME								
PREPARED BY								
DESIGNATION								
DATE								
Test Executed by								
S.NO	TEST CASE ID	Component/Module	Priority	Description	Pre-requisites	Test Steps		
1	GoogleSearch_1	Search_Bar_Module	High	Verify that when a user writes a search term and presses enter, search results should be displayed	Browser is launched	1. Write the http://goo in the brow URL bar ar enter. 2. Once google.co launched, the search "Apple" ir google sea 3. Press en		

Main fields of Example 2 Case:

TestCaseId – This field uniquely identifies a test case. It is mapped with automation scripts (if any) to keep a track of automation status. The same field can be used for mapping with the test scenarios for generating a traceability matrix. E.g. – GoogleSearch_1

Component/Module – This field specifies the specific component or module that the test case belongs to. E.g. – Search_Bar_Module

Priority – This field is used to specify the priority of the test case. Normally the conventional followed for specifying the priority is either High, Medium, Low or P0, P1, P3, P3, etc with P0 being the most critical.

Description – In this field describe the test case in brief. E.g. – Verify that when a user writes a search term and presses enter, search results should be displayed.

Pre-requisites – In this field specify the conditions or steps that must be followed before the test steps executions. E.g. – Browser is launched.

Test Steps – In this field, we mention each and every step for performing the test case. The test steps should be clear and unambiguous. E.g. Write the URL – http://google.com in the browser's URL bar and press enter.

Once google.com is launched, write the search term – "Apple" in the google search bar.

Press enter.

TestData – In this field, we specify the test data used in the test steps. E.g. in the above test step example we could use the search term-“apple” as test data.

Expected Result – This step marks the expected result after the test step execution. This used to assert the test case. E.g. – search results related to ‘apple’ should be displayed.

Actual Result – In this step, we specify the actual result after the test step execution. E.g. – search results with ‘apple’ keyword were displayed.

Status/Test Result – In this step, we mark the test case as pass or fail based on the expected and actual result. Possible values can be – Pass, Fail, Not executed.

Test Executed by – In this field we specify the tester’s name who executed the test case and marked the test case as pass or fail.

Apart from these mandatory fields, there are many optional fields that can be added as the organization or application’s need like

Automation status – for marking test as automated or manual,

TestScenarioId – for mapping test case with its test scenario,

AfterTest step – for specifying any step required to be executed after performing the test case and

TestType – to specify if the test is applicable for Regression, Sanity, Smoke, etc and

Defected – id of the defect launched in any of the defect management tools, etc.

LIST OF WEB APPLICATION TESTING EXAMPLE TEST CASES/SCENARIOS.

Assumptions: Assume that your application supports the following functionalities

- Forms with various fields
- Child windows
- The application interacts with the database
- Various search filter criteria and display results
- Image upload
- Send email functionality
- Data export functionality

EXERCISE: GENERAL TEST SCENARIOS

1. All mandatory fields should be validated and indicated by an asterisk (*) symbol.
2. Validation error messages should be displayed properly in the correct position.
3. All error messages should be displayed in the same CSS style (**For Example,** using red color)
4. General confirmation messages should be displayed using CSS style other than error

messages style (**For Example,** using green color)
5. Tooltips text should be meaningful.
6. Drop-down fields should have the first entry as blank or text like ‘Select'.
7. ‘Delete functionality' for any record on a page should ask for a confirmation.
8. Select/deselect all records option should be provided if page supports record add/delete/update functionality
9. Amount values should be displayed with correct currency symbols.
10. Default page sorting should be provided.
11. Reset button functionality should set default values for all fields.
12. All numeric values should be formatted properly.
13. Input fields should be checked for the max field value. Input values greater than the specified max limit should not be accepted or stored in the database.
14. Check all input fields for special characters.
15. Field labels should be standard e.g. field accepting user's first name should be labeled properly as ‘First Name'.
16. Check page sorting functionality after add/edit/delete operations on any record.
17. Check for timeout functionality. Timeout values should be configurable. Check application behavior after the operation timeout.
18. Check cookies used in an application.
19. Check if downloadable files are pointing to the correct file paths.
20. All resource keys should be configurable in config files or database instead of hard coding.
21. Standard conventions should be followed throughout for naming resource keys.
22. Validate markup for all web pages (validate HTML and CSS for syntax errors) to make sure it is compliant with the standards.
23. Application crash or unavailable pages should be redirected to the error page.
24. Check the text on all pages for spelling and grammatical errors.
25. Check numeric input fields with character input values. A proper validation message should appear.
26. Check for negative numbers if allowed for numeric fields.
27. Check the number of fields with decimal number values.
28. Check the functionality of buttons available on all pages.
29. The user should not be able to submit a page twice by pressing the submit button in quick succession.
30. Divide by zero errors should be handled for any calculations.
31. Input data with the first and last position blank should be handled correctly.

EXERCISE: GUI AND USABILITY TEST SCENARIOS

1. All fields on a page (**For Example,** text box, radio options, drop-down lists) should be aligned properly.
2. Numeric values should be justified correctly unless specified otherwise.
3. Enough space should be provided between field labels, columns, rows, error messages, etc.
4. The scrollbar should be enabled only when necessary.
5. Font size, style, and color for headline, description text, labels, infield data, and grid info should be standard as specified in SRS.
6. The description text box should be multi-lined.

7. Disabled fields should be greyed out and users should not be able to set focus on these fields.
8. Upon click of an input text field, the mouse arrow pointer should get changed to the cursor.
9. The user should not be able to type in drop-down select lists.
10. Information filled by users should remain intact when there is an error message on page submit. The user should be able to submit the form again by correcting the errors.
11. Check if proper field labels are used in error messages.
12. Drop-down field values should be displayed in defined sort order.
13. Tab and Shift+Tab order should work properly.
14. Default radio options should be pre-selected on the page load.
15. Field-specific and page-level help messages should be available.
16. Check if the correct fields are highlighted in case of errors.
17. Check if the drop-down list options are readable and not truncated due to field size limits.
18. All buttons on a page should be accessible by keyboard shortcuts and the user should be able to perform all operations using a keyboard.
19. Check all pages for broken images.
20. Check all pages for broken links.
21. All pages should have a title.
22. Confirmation messages should be displayed before performing any update or delete operation.
23. Hourglass should be displayed when the application is busy.
24. Page text should be left-justified.
25. The user should be able to select only one radio option and any combination for checkboxes.

EXERCISE: TEST SCENARIOS FOR FILTER CRITERIA

1. The user should be able to filter results using all parameters on the page.
2. Refine search functionality should load the search page with all user-selected search parameters.
3. When there are at least one filter criteria required to perform the search operation, make sure the proper error message is displayed when the user submits the page without selecting any filter criteria.
4. When at least one filter criteria selection is not compulsory, the user should be able to submit the page and the default search criteria should get used to query results.
5. Proper validation messages should be displayed for all invalid values for filter criteria.

EXERCISE: TEST SCENARIOS FOR RESULT GRID

1. Page loading symbol should be displayed when it's taking more than default time to load the result page.
2. Check if all the search parameters are used to fetch data shown on the result grid.
3. The total number of results should be displayed in the result grid.
4. Search criteria used for searching should be displayed in the result grid.
5. Result grid values should be sorted by default column.
6. Sorted columns should be displayed with a sort icon.
7. Result grids should include all the specified columns with correct values.
8. Ascending and descending sorting functionality should work for columns supported by

data sorting.
9. Result grids should be displayed with proper column and row spacing.
10. Pagination should be enabled when there are more results than the default result count per page.
11. Check for Next, Previous, First and Last page pagination functionality.
12. Duplicate records should not be displayed in the result grid.
13. Check if all the columns are visible and a horizontal scrollbar is enabled if necessary.
14. Check the data for dynamic columns (columns whose values are calculated dynamically based on the other column values).
15. For result grids showing reports check 'Totals' row and verify the total for every column.
16. For result grids showing reports check 'Totals' row data when pagination is enabled and the user gets navigated to the next page.
17. Check if proper symbols are used for displaying column values e.g. % symbol should be displayed for percentage calculation.
18. Check result grid data to know if the date range is enabled.

EXERCISE: TEST SCENARIOS FOR A WINDOW

1. Check if the default window size is correct.
2. Check if the child window size is correct.
3. Check if there is any field on the page with default focus (in general, the focus should be set on the first input field of the screen).
4. Check if child windows are getting closed on closing parent/opener window.
5. If the child window is opened, the user should not be able to use or update any field in the background or parent window
6. Check window minimize, maximize, and close functionality.
7. Check if the window is re-sizable.
8. Check scroll bar functionality for parent and child windows.
9. Check cancel button functionality for the child window.

EXERCISE: DATABASE TESTING TEST SCENARIOS

1. Check if correct data is getting saved in the database upon a successful page submit.
2. Check values for columns that are not accepting null values.
3. Check for data integrity. Data should be stored in single or multiple tables based on the design.
4. Index names should be given as per the standards e.g.
IND_<Tablename>_<ColumnName>
5. Tables should have a primary key column.
6. Table columns should have description information available (except for audit columns like created date, created by, etc.)
7. For every database add/update operation log should be added.
8. Required table indexes should be created.
9. Check if data is committed to the database only when the operation is successfully completed.
10. Data should be rolled back in case of failed transactions.
11. Database name should be given as per the application type i.e. test, UAT, sandbox, live (though this is not a standard it is helpful for database maintenance)
12. Database logical names should be given according to the database name (again this is

not standard but helpful for DB maintenance).
13. Stored procedures should not be named with a prefix “sp_”
14. Check if values for table audit columns (like created date, created by, updated, updated by, is deleted, deleted data, deleted by, etc.) are populated properly.
15. Check if input data is not truncated while saving. Field length shown to the user on the page and in database schema should be the same.
16. Check numeric fields with minimum, maximum, and float values.
17. Check numeric fields with negative values (for both acceptance and non-acceptance).
18. Check if the radio button and drop-down list options are saved correctly in the database.
19. Check if the database fields are designed with the correct data type and data length.
20. Check if all the table constraints like a Primary key, Foreign key, etc. are implemented correctly.
21. Test stored procedures and triggers with sample input data.
22. Input field leading and trailing spaces should be truncated before committing data to the database.
23. Null values should not be allowed for the Primary key column.

EXERCISE: TEST SCENARIOS FOR IMAGE UPLOAD FUNCTIONALITY

(Also applicable for other file upload functionality)
1. Check for uploaded image path.
2. Check image upload and change functionality.
3. Check image upload functionality with image files of different extensions (**For Example,** JPEG, PNG, BMP, etc.)
4. Check image upload functionality with images having space or any other allowed special character in the file name.
5. Check duplicate name image upload.
6. Check image upload with image size greater than the max allowed size. The Proper error message should be displayed.
7. Check image upload functionality with file types other than images (**For Example,** txt, doc, pdf, exe, etc.). A proper error message should be displayed.
8. Check if images of specified height and width (if defined) are accepted otherwise rejected.
9. The image upload progress bar should appear for large size images.
10. Check if the cancel button functionality is working in between the upload process.
11. Check if file selection dialog shows only supported files listed.
12. Check multiple images upload functionality.
13. Check image quality after upload. Image quality should not be changed after upload.
14. Check if the user is able to use/view the uploaded images.

EXERCISE: TEST SCENARIOS FOR SENDING EMAILS

(Test cases for composing or validating emails are not included here)
(Make sure to use dummy email addresses before executing email related tests)
1. The email template should use standard CSS for all emails.
2. Email addresses should be validated before sending emails.
3. Special characters in the email body template should be handled properly.
4. Language-specific characters (**For Example,** Russian, Chinese or German language characters) should be handled properly in the email body template.
5. Email subject should not be blank.

6. Placeholder fields used in the email template should be replaced with actual values e.g. {Firstname} {Lastname} should be replaced with an individual's first and last name properly for all the recipients.
7. If reports with dynamic values are included in the email body and report data should be calculated correctly.
8. Email sender name should not be blank.
9. Emails should be checked in different email clients like Outlook, Gmail, Hotmail, Yahoo! mail, etc.
10. Check to send email functionality using TO, CC and BCC fields.
11. Check plain text emails.
12. Check HTML format emails.
13. Check email header and footer for company logo, privacy policy, and other links.
14. Check emails with attachments.
15. Check to send email functionality to single, multiple or distribution list recipients.
16. Check if a reply to the email address is correct.
17. Check to send the high volume of emails.

EXERCISE: TEST SCENARIOS FOR EXCEL EXPORT FUNCTIONALITY

1. The file should get exported in the proper file extension.
2. The file name for the exported Excel file should be as per the standards, **For Example,** if the file name is using the timestamp, it should get replaced properly with an actual timestamp at the time of exporting the file.
3. Check for date format if exported Excel file contains the date columns.
4. Check number formatting for numeric or currency values. Formatting should be the same as shown on the page.
5. The exported file should have columns with proper column names.
6. Default page sorting should be carried in the exported file as well.
7. Excel file data should be formatted properly with header and footer text, date, page numbers, etc. values for all pages.
8. Check if the data displayed on a page and exported Excel file is the same.
9. Check export functionality when pagination is enabled.
10. Check if the export button is showing proper icon according to the exported file type, **For Example,** Excel file icon for xls files
11. Check export functionality for files with very large size.
12. Check export functionality for pages containing special characters. Check if these special characters are exported properly in the Excel file.

EXERCISE: PERFORMANCE TESTING TEST SCENARIOS

1. Check if the page load time is within the acceptable range.
2. Check the page load on slow connections.
3. Check the response time for any action under a light, normal, moderate, and heavy load conditions.
4. Check the performance of database stored procedures and triggers.
5. Check the database query execution time.
6. Check for load testing of the application.
7. Check for the Stress testing of the application.
8. Check CPU and memory usage under peak load conditions.

EXERCISE: SECURITY TESTING TEST SCENARIOS

1. Check for SQL injection attacks.
2. Secure pages should use the HTTPS protocol.
3. Page crash should not reveal application or server info. The error page should be displayed for this.
4. Escape special characters in the input.
5. Error messages should not reveal any sensitive information.
6. All credentials should be transferred over an encrypted channel.
7. Test password security and password policy enforcement.
8. Check application logout functionality.
9. Check for Brute Force Attacks.
10. Cookie information should be stored in encrypted format only.
11. Check session cookie duration and session termination after timeout or logout.
11. Session tokens should be transmitted over a secured channel.
13. The password should not be stored in cookies.
14. Test for Denial of Service attacks.
15. Test for memory leakage.
16. Test unauthorized application access by manipulating variable values in the browser address bar.
17. Test file extension handing so that exe files are not uploaded and executed on the server.
18. Sensitive fields like passwords and credit card information should not have to autocomplete enabled.
19. File upload functionality should use file type restrictions and also anti-virus for scanning uploaded files.
20. Check if directory listing is prohibited.
21. Passwords and other sensitive fields should be masked while typing.
22. Check if forgot password functionality is secured with features like temporary password expiry after specified hours and security question is asked before changing or requesting a new password.
23. Verify CAPTCHA functionality.
24. Check if important events are logged in log files.
25. Check if access privileges are implemented correctly.

11. SOFTWARE TEST ESTIMATION TECHNIQUES: STEP BY STEP GUIDE

For the success of any project test estimation and proper execution is equally important as the development cycle. Sticking to the estimation is very important to build a good reputation with the client. Experience plays a major role in estimating “Software Testing Efforts”. Working on varied projects helps to prepare an accurate estimation of the testing cycle. Obviously one cannot just blindly put some number of days for any testing task. Test estimation should be realistic and accurate.

BRIEF DESCRIPTION OF THE TEST ESTIMATION PROCESS

“Estimation is the process of finding an estimate, or approximation, which is a value that is usable for some purpose even if input data may be incomplete, uncertain, or unstable.”

We all come across different tasks and duties and deadlines throughout our lives as professionals, now there are two approaches to find a solution to a problem. A first approach is a reactive approach whereby we try to find a solution to the problem at hand only after it arrives. In the second approach which can be called a Proactive Approach whereby we first prepare ourselves well before the problem arrives with our past experiences and then with our past experience, we try to find a solution to the challenge when it arrives. Estimation can thus be considered as a technique that is applied when we take a proactive approach to the problem. Thus Estimation can be used to predict how much effort with respect to time and cost would be required to complete a defined task. Once the testing team is able to make an estimate of the problem at hand then it is easier for them to come up with a solution that would be optimum to the problem at hand. The practice of estimation can be defined then more formally as an approximate computation of the probable cost of a piece of work.

THE BASIC PREREQUISITES OF THE TEST ESTIMATION PROCESS

#1) Insights gathered from working with past experience: It is always a good practice to spend some time, recalling past projects which posed challenges similar to the current endeavor at hand.

#2) The available documents or artifacts: The test management repository tools come in handy in these types of scenarios as they store the requirement and clarification documents. These documents can be referred by the testing team to clearly define the scope of the project.

#3) Assumptions about the type of work: Past working experience helps in making assumptions about the project. This is where hiring experienced professionals matters most.

The testing managers can pick up the brains of these people for delivering the desired results.

#4) Calculation of potential risks and threats: The testing team also needs to visualize the potential risks and threats and pitfalls which lie may lie for the team in the future.

#5) Determining whether the documents have been baselined: The testing team also needs to determine if the requirements have been baselined or not. If the documents are not baselined then it is important to determine the frequency of the changes.

#6) All responsibilities and dependencies should be clear: The organization should clearly define the roles and responsibilities for all the persons who would be performing the estimation process.

#7) Documentation and tracking of the estimation records: All the relevant information to the estimation process should be documented.

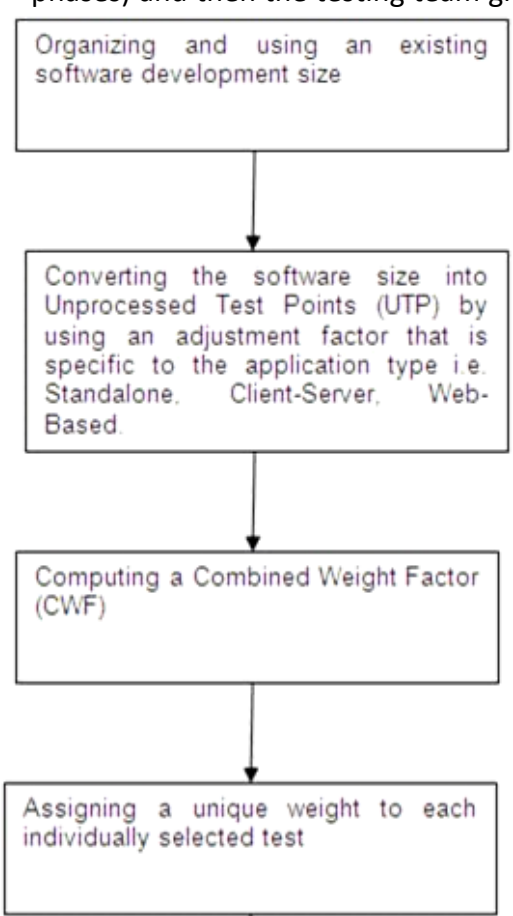
#8) Activities which are required to be performed during the test estimation process

- Organize the team that would perform estimations
- Decompose the project into project phases and subsequent constituent activities
- Compute the estimation based upon previous projects and professional experience
- Prioritize the possible threats and come up with the approaches to mitigate those risks
- Review and document the relevant part of the work
- Submit the work to the relevant stakeholders
- Most Prominent Test Estimation Techniques
- Some of the most important techniques for test estimation are:
- Test point estimation
- Work-phase based estimation
- Use case point estimation

HOW AND WHERE WE USE THESE TECHNIQUES:

#1) Test Point estimation is a simple and easily understandable estimation technique that is widely used across the software testing spectrum. Iterative phases and simplicity are the most important features of this particular technique.

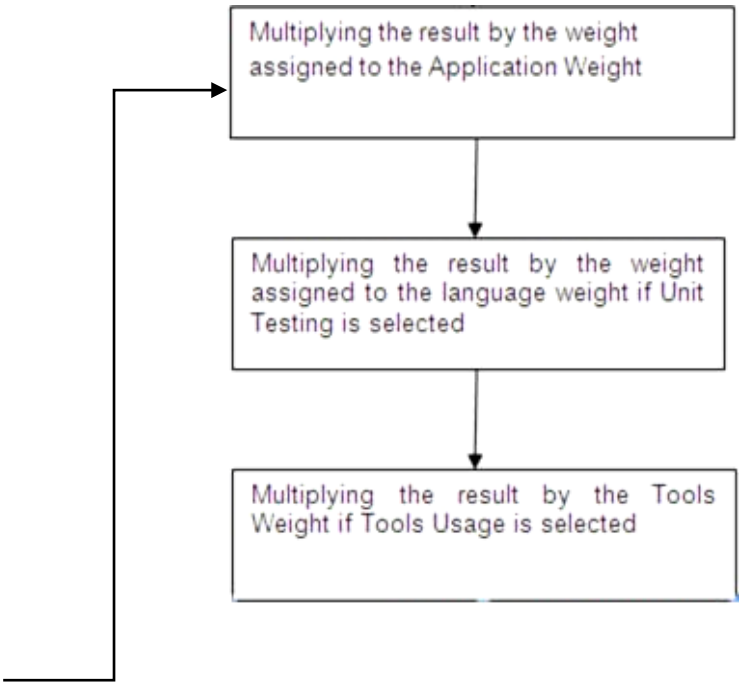
#2) Work-phase based estimation is the estimation technique which is used whereby a guess estimate is made on a particular phase (normally the shortest and simplest of the phases) and then the testing team gradually adds on other phases into the initial estimation and finally comes up with an appropriate estimation.



#3) Use-Case Point estimation technique is the estimation on the use cases where the unadjusted actor weights and unadjusted use case weights are used to determine the software testing estimation.

DETAILS OF THE TEST POINT ESTIMATION TECHNIQUE

The test point estimation technique is done by following the listed steps:



Unprocessed Test Points are multiplied by CWF to obtain the testing size in Test Point's Size.

The Productivity Factor indicates the amount of time for a test engineer to complete the testing of one Test Point.

Testing Effort in Person Hours is computed by multiplying the Test Point size by the Productivity factor.

For the computation of the test point estimation technique, we consider the following variables.

Test requirement complexity

Complexity Type	Complexity
Simple	< 2 transactions
Average	3-6 transactions
Complex	> 6 transactions

Interface with other requirements

Complexity Type	Interface with other requirements
Simple	0
Average	< 3
Complex	>3

Total number of verification points

Complexity Type	Total number of verification points
Simple	<2
Average	3-8
Complex	>8

Baseline test data

Complexity Type	Baseline Test Data
Simple	Not required
Average	Required
Complex	Required

We then need to consider weight vectors for each of the data variables and organize them in the following manner.

Factor	Factor Weight	Complexity Weight
Domain Weightage	(0-10)	(0-3)
Horizontal Weightage	(0-10)	(0-3)
Test Data Preparation	(0-10)	(0-3)
Multilingual Support	(0-10)	(0-3)
Environment setup and Network Latency	(0-10)	(0-3)
Requirements Stability	(0-10)	(0-3)
Integration with other hardware devices	(0-10)	(0-3)
Configuration management	(0-10)	(0-3)

Adjustment factor = Average of (product of complexity weight and factor weight) / 30

Adjustment Test Point for Test case design = Total Test Point X (1 + Adjustment factor for Test Case design)

Adjusted Test Point for Test case execution = Total Test Point X (1 + Adjustment factor for Test Case execution)

Total Test Point (normalized) X (1 + Adjustment factor for Test Case design/execution) = Adjusted Test Point for Test Case design/execution

Total effort in Person Hours (PH) = Number of Normalized Test points / Productivity (in Normalized Test points per Person Hours)

TEST ESTIMATION EXERCISE:

Let us try to apply the above formulation into a practical use.

Suppose we end up with a test requirement whereby we have 5 test scenarios to test.

Now say Test scenario 1 has got 5 test expected results, test scenario 2 6 test expected results, test scenario 3 only 2 test expected results, test scenario 4 9 test expected results, test scenario 5 also 9 test expected results, respectively.

So we classify the test scenarios in three classes i.e. complex, simple and moderate based upon the total number of expected results present in these three classes.

Complex classes will have more than 7 expected results whereas the simple ones will consist of less than 5 expected results and the moderate scenarios would consist of between 4 to 7 expected results.

We thus classify test scenario 1 and test scenario 2 as moderate scenarios, scenario 5 and scenario 6 as complex ones and test scenario 3 as simple.

We now will apply test points to all these scenarios. We apply 5 test points for complex classes, 3 for moderate ones and 2 for the simple scenarios.

We multiply the assumed test points with the total number of expected results in all these test scenarios. So we end up with the following approximations.

*Scenario 1: 3 test points * 5 test expected results = Adjusted test points = 25*
*Scenario 2: 3 test points * 6 test expected results = Adjusted test points = 30*
*Scenario 3: 2 test points * 2 test expected results = Adjusted test points = 4*
*Scenario 4: 5 test points * 9 test expected results = Adjusted test points = 45*
*Scenario 5: 5 test points * 9 test expected results = Adjusted test points = 45*

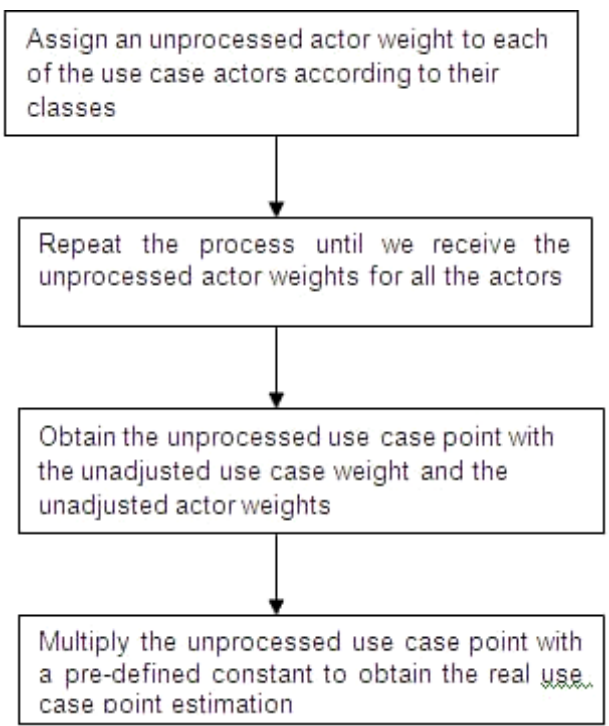
So considering that we need to apply to say 5 Person Hours for each adjusted test point we end up getting the following approximate result.

*Test Scenario 1: 25 adjusted test points * 5 Person Hours = 125 Person Hours*
*Test Scenario 2: 30 adjusted test points * 5 Person Hours = 150 Person Hours*
*Test Scenario 3: 4 adjusted test points * 5 Person Hours= 20 Person Hours*
*Test Scenario 4: 45 adjusted test points * 5 Person Hours = 225 Person Hours*
*Test Scenario 5: 45 adjusted test points * 5 Person Hours = 225 Person Hours*

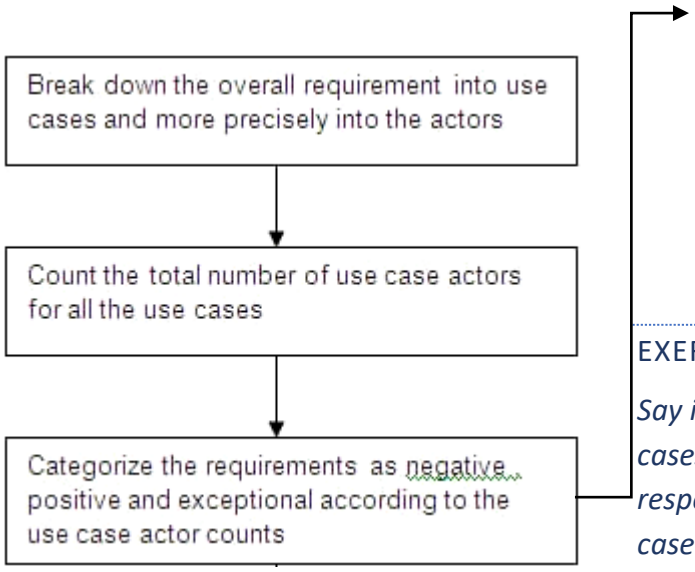
So total approximated person-hours is: 745 Person Hours

USE CASE POINT ESTIMATION METHOD

Use-Case Point Method is based on the use cases where we calculate the overall test estimation effort based on the use cases or the requirements.



Here is the detailed process of the Use case point estimation method:



EXERCISE:

Say in a particular requirement we have 5 use cases, use case 1, use case 2,…, use case 5 respectively. Now let us consider that use case 1 consists of 6 actors, use case 2 consists of 15 actors, use cases 3, 4 and 5, 3, 4 and 5 actors respectively.

We consider any use case which involves the total number of actors as less than 5 as negative, any use case with the total number of actors is equal to or more than 5 and less than or equal to 10 as positive and any use case with more than 10 actors as exceptional.

We decide to assign 2 points to the exceptional use cases, 1 to the positive ones and -1 for the negative ones.

Thus we categorize the Use cases 1 and 5 as positive, use case 2 as exceptional and use case 3, 4 as negative respectively based on our above-stated assumptions.

*So the Unprocessed actor weights = Use case 1 = (total number of actors) 5 * 1(the assigned point) = 5. Similarly*

*Use case 2 = 15 * 2 = 30 .*

Repeating the process for rest of the use cases we receive the Unprocessed actor weights = 33

Unprocessed use case weight = total no. of use cases = 5

Unprocessed use case point = Unadjusted actor weights + Unadjusted use case weight = 33 + 5 = 38

*Processed use case point = 38 * [0.65+ (0.01 * 50] = 26.7 or 28 Person Hours approximately*

WORK-PHASE BREAKDOWN TECHNIQUE

The work phase breakdown technique can be described in the following steps.

- Break down the overall work into phases.
- Start with the simplest phase and assign an approximate estimation value to it.
- Then proceed with identifying the next possible phase which could be commenced once this phase completes.
- Derive a possible set of approximation values that could be applied to this phase and choose the maximum value amongst all the derived approximation values.
- Sum up the approximated estimation value by adding the current phase effort estimation value to the already existing value.
- Continue steps 3 to 5 until all the phases identified in the first step are exhausted.
- Accept the final approximate estimate value as the ultimate.

EXERCISE:

Suppose in a requirement there are 5 required phases. So in the initial phase 1 we assume that total efforts needed are 35 person-hours and then we commence the next phase 2 for which we have 4 comparative assumptions of 35, 45, 55 and 65 respectively.

So we consider the 65 person-hour which is the maximum value here. In phase 3,4 ,5 we come up with estimates (12,33,43 ,54) , (15, 10, 7 , 8) and (2 , 16 , 5 ,13) respectively. By applying the said principle we end up with 185 Person Hours respectively.

FACTORS AFFECTING SOFTWARE TEST ESTIMATION, AND GENERAL TIPS TO ESTIMATE ACCURATELY:

#1) Think of Some Buffer Time: The estimation should include some buffer. But do not add a buffer, which is not realistic. Having a buffer in the estimation enables to cope with any delays that may occur. Having a buffer also helps to ensure maximum test coverage.

#2) Consider the Bug Cycle: The test estimation also includes the bug cycle. The actual test cycle may take more days than estimated. To avoid this, we should consider the fact that the test cycle depends on the stability of the build. If the build is not stable, then developers may need more time to fix and obviously, the testing cycle gets extended automatically.

#3) Availability of All the Resources for Estimated Period: The test estimation should consider all the leaves planned by the team members (typically long leaves) in the next few weeks or next few months. This will ensure that the estimations are realistic.

The estimation should consider some fixed number of resources for a test cycle. If the number of resources reduces then the estimation should be re-visited and updated accordingly.

#4) Can We Do Parallel Testing?: Do you have some previous versions of the same product so that you can compare the output? If yes, then this can make your testing task a bit easier. You should think about the estimation based on your product version.

#5) Estimations Can Go Wrong: So re-visit the estimations frequently in initial stages before you commit it. In the early stages, we should frequently re-visit the test estimations and make a modification if needed. We should not extend the estimation once we freeze it unless there are major changes in requirements.

#6) Think of Your Past Experience to Make Judgments!: Experiences from past projects play a vital role while preparing time estimates. We can try to avoid all the difficulties or issues that were faced in past projects. We can analyze how the previous estimates were and how much they helped to deliver the product on time.

#7) Consider the Scope of Project: Know what is the end objective of the project and list of all final deliverables. Factors to be considered for small and large projects differ a lot.

A Large project typically includes setting up a testbed, generating test data, test scripts, etc. Hence the estimations should be based on all these factors. Whereas in small projects, typically the test cycle includes test cases writing, execution and regression.

#8) Are You Going to Perform Load Testing?: If you need to put considerable time on performance testing then estimate accordingly. Estimations for projects, which involve load testing, should be considered differently.

#9) Do You Know Your Team?: If you know the strengths and weaknesses of individuals working in your team then you can estimate testing tasks more precisely. While estimating one should consider the fact that all resources may not yield the same productivity level. Some people can execute faster compared to others. Though this is not a major factor, it adds up to the total delay in deliverables.

12. HOW TO WRITE A TEST PLAN

A TEST PLAN is a document describing software testing scope and activities. It is the basis for formally testing any software/product in a project.

Definition: A document describing the scope, approach, resources and schedule of intended test activities. It identifies amongst others test items, the features to be tested, the testing tasks, who will do each task, degree of tester independence, the test environment, the test design techniques and entry and exit criteria to be used, and the rationale for their choice,and any risks requiring contingency planning. It is a record of the test planning process. There can be a Master test plan (A test plan that typically addresses multiple test levels) and a Phase test plan (A test plan that typically addresses one test phase).

TEST PLAN TYPES

- Master Test Plan: A single high-level test plan for a project/product that unifies all other test plans.
- Testing Level Specific Test Plans: Plans for each level of testing.
- Unit Test Plan
- Integration Test Plan
- System Test Plan
- Acceptance Test Plan
- Testing Type Specific Test Plans: Plans for major types of testing like Performance Test Plan and Security Test Plan.

TEST PLAN TEMPLATE

The format and content of a software test plan vary depending on the processes, standards, and test management tools being implemented. Nevertheless, the following format, which is based on IEEE standard for software test documentation, provides a summary of what a test plan can/should contain.

Test Plan Identifier: Provide a unique identifier for the document. (Adhere to the Configuration Management System if you have one.)

Introduction:

- Provide an overview of the test plan.
- Specify the goals/objectives.
- Specify any constraints.

References:

- List the related documents, with links to them if available, including the following:
- Project Plan
- Configuration Management Plan

Test Items:

- List the test items (software/products) and their versions.
- Features to be Tested:
- List the features of the software/product to be tested.
- Provide references to the Requirements and/or Design specifications of the features to be tested

Features Not to Be Tested:

- List the features of the software/product which will not be tested.

- Specify the reasons these features won't be tested.

Approach:

- Mention the overall approach to testing.
- Specify the testing levels [if it's a Master Test Plan], the testing types, and the testing methods [Manual/Automated; White Box/Black Box/Gray Box]
- Item Pass/Fail Criteria:
- Specify the criteria that will be used to determine whether each test item (software/product) has passed or failed testing.
- Suspension Criteria and Resumption Requirements:
- Specify criteria to be used to suspend the testing activity.
- Specify testing activities which must be redone when testing is resumed.

Test Deliverables:

List test deliverables, and links to them if available, including the following:

- Test Plan (this document itself)
- Test Cases
- Test Scripts
- Defect/Enhancement Logs
- Test Reports

Test Environment:

- Specify the properties of test environment: hardware, software, network etc.
- List any testing or related tools.

Estimate: Provide a summary of test estimates (cost or effort) and/or provide a link to the detailed estimation.

Schedule: Provide a summary of the schedule, specifying key test milestones, and/or provide a link to the detailed schedule.

Staffing and Training Needs:

- Specify staffing needs by role and required skills.
- Identify training that is necessary to provide those skills, if not already acquired.
- Responsibilities:
- List the responsibilities of each team/role/individual.

Risks:

- List the risks that have been identified.
- Specify the mitigation plan and the contingency plan for each risk.
- Assumptions and Dependencies:

- List the assumptions that have been made during the preparation of this plan.
- List the dependencies.

Approvals:

- Specify the names and roles of all persons who must approve the plan.
- Provide space for signatures and dates. (If the document is to be printed.)

TEST PLAN GUIDELINES

- Make the plan concise. Avoid redundancy and superfluousness. If you think you do not need a section that has been mentioned in the template above, go ahead and delete that section in your test plan.
- Be specific. For example, when you specify an operating system as a property of a test environment, mention the OS Edition/Version as well, not just the OS Name.
- Make use of lists and tables wherever possible. Avoid lengthy paragraphs.
- Have the test plan reviewed a number of times prior to baselining it or sending it for approval. The quality of your test plan speaks volumes about the quality of the testing you or your team is going to perform.
- Update the plan as and when necessary. An out-dated and unused document stinks and is worse than not having the document in the first place.

EXAMPLE TEST PLAN:

XYZ bank

Detailed Test Plan - UIS

XYZ System

This copy printed on:	February 11, 2021
Document Last Modification date:	May 19 2016
Version:	0.1
Status:	<Draft >
Revision:	1

Document Owner: XYZ BANK PCL.

Document Author: XYXY

Document Reviewer: <Reviewer/Manager>

Document History

This is a snapshot of an on-line document. Paper copies are valid only on the day they are printed. Refer to the author if you are in any doubt about the currency of this document.

This is document template format when you copy to use for practice please remove yellow highlight and replace with your own information.

Revision History

Version Number	Revision Date	Summary of Changes	Changed by
V 0.1	08 Nov 2015	Initial Version	

Reviewer

Name	Position/ Department	Version	Review Date
(Name-Surname)	(Position / Department)	9.9	DD/MM/YYYY

Approvals

Name	Position/ Department	Version	Approve Date
Name-Surname)	(Position / Department)	9.9	DD/MM/YYYY

Distribution

This document is available to:

☐ Entire Project

☐ Restricted to the following team members

Name	Title
(name)	(title)

List of Table

Introduction

Background

The Detailed Test Plan (DTP) contains a detailed and executable strategy for conducting. It defines the detailed testing objective specific to a particular system, the testing approach, test environment, test conditions, and the test plan.

Target is to concentrate on internal and external fraud detection. The primary

stage will concentrate on the detection of internal fraud at the retail and private

branch operations and at the activity of staff accounts that defined from XYZ bank.

Testing Objectives

The objectives of testing are as follows:

• Implement a world-class fraud detection system that will be scalable and expandable and will assist the Bank compact and significantly reduce both internal and external fraud risk.

• Develop a fraud detection capability that allows the management of multiple fraud risk types through a common platform.

• Provide a robust fraud detection platform that can be used across multiple enterprises across the XYZ bank.

Document Audience

Role	Name	Email/ Telephone	Organization
Product Owner			
Project Manager			
PMO			
QA			
Functional Team Lead			
BA			
Business Unit			
Development			

Role	Name	Email/ Telephone	Organization
Team Lead			
Infrastructure Team Lead			
Infrastructure Team			
Test Lead			
Test Team			
Deployment Team Lead			
Deployment Team			

* Remove the role which may not include in your project charter. * If document change version and the stakeholder name is changed, should modify the name as well..

References

This document is based on and refers to the following documents:

Table 1: Reference

Document Name	Author	Version	Update date
1. <Master Test plan>	<Name of last update person>	<V.9.9>	<Latest update date>
2. Bank Specification		V.0.1	February 11,

			2021
3. <RTM : Requirement Traceability Matrix>	<Name of last update person>	<V.9.9>	<Latest update date>

Definition and Acronyms

This section provides information regarding the Acronyms and terminology specifically used in this document.

Table 2: Definitions and Acronyms

Acronym	Definition
CD	Cash Deposit
CW	Cash Withdrawal

Testable Items

In Scope

This test activity focuses on the following:

1. Perform investigations based on business rule.

2.

3.

Functional Scope

To list all functional area and description that will be in scope of testing.

Table 3: Functional

Functional Area	Functional Sub Area	Description
BR001	BR001_1	Detailed description
BR002	BR002_1	Detailed description
BR003	BR003_1	Detailed description

Non Functional Scope

To list all non functional area and description that will be in scope of testing.

Table 4: Non Functional

Non Functional Requirement	Description
NFR-001	Detailed description
NFR-002	Detailed description

Out of Scope

[It is important to clearly define (at a high level) all of the testable components of the solution that will not be tested. These should include infrastructure, functional subsets, non-functional requirements, and software modules. Specific testing activities (such as load and performance testing, penetration tests, etc) should also be listed. Brief reasoning behind why the items have been de-scoped should be included].

Table 5: Out of Scope Content

Item	Description
1	<High level describe the item of out scope>
2	<High level describe the item of out scope>
3	<High level describe the item of out scope>

Testing Exclusions

[A test exclusion is an element of the SUT that has not been de-scoped but which will not be tested by this plan due to (usually) a logistical issue. These may include activities such as report/letters distributed by fax/email/sms and may reference the test phase/level that will be responsible for conducting this testing].

Detailed Test Approach

All testing conducted by, or for, the KT Program complies with the Master Test Strategy. This DTP has been created to define the test activities documented in the Master Test Plan

[Insert Release Name and Test Activity] will be tested using the following approach:

[List (in detail) the key elements that make up the strategy you will use to deliver the required test objectives. Wherever possible link specific activities to the relevant test objective. Change/edit the accompanying descriptions to suit the individual release].

Naming Conventions for Function Groups and Functions

[Use the Configuration Management naming conventions to define the structure used for the test project nomenclature. If HP Quality Center is to be used ensure the nomenclature rules also align with the requirements defined for it. The Test case ID should be agreed with the naming convention.

Rename this section as required:

… for Function Groups and Functions

… for Scenarios and Usecases

etc]

Test Case Design

Extract testable requirements from the requirement specifications and design test conditions which accurately reflect the functional enhancements and changes.

[Add design elements/requirements specific to this test activity].

Identify test approach about test case design before identify the detail of test case test script in next step. To declare what is the concern item of each module or functional/non functional area and what the test case design is for detect the defect. The test technique should be state for this section. XYZ bank test coverage matrix should be mention and state that how to ensure the project will follow this guideline.

Test design should mention about the group of regression impact. If the most of the defect occur on the area which test case design should be re-test. And the regression test approach should be mention as well.

Test Scheduling

Identify test schedule for <Test level> base on <project master plan version>. The high level plan for testing preparation, test execution, defect fixing, and data test preparation period are identified as below. <The format of schedule can demonstrate in structure of table of calendar or schedule in Microsoft project>

[For complex test schedules reference any external tools or systems you will be using/developing to support the scheduling. Include specific flags or nomenclature that will be used to identify/classify test scripts and test cases that have to be executed at a specific point in the schedule. (pre or post batch, day one, after script xyz has executed, etc]

Figure: Test schedule

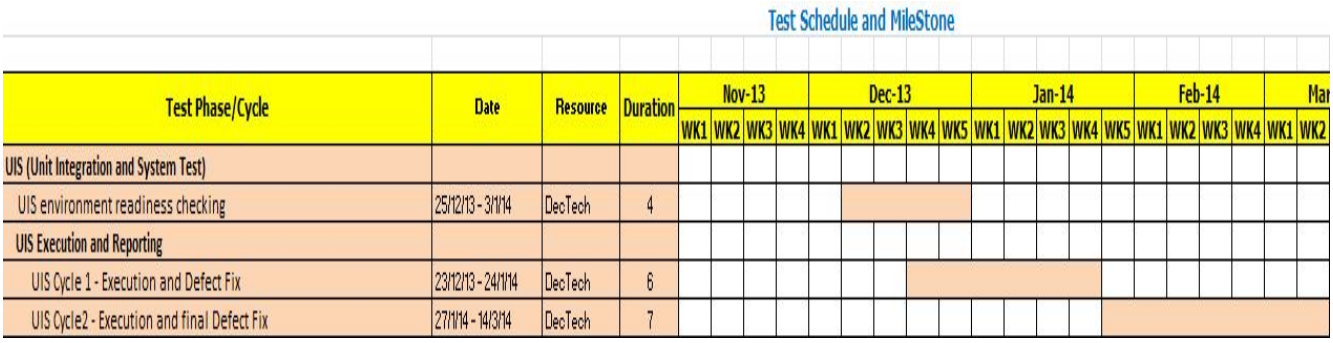


Table 6: Example test cycle

Batch Run #	Function to be Executed	Cycle Date
Batch Run 1 (Daily)	Account Enquiries New Account Set-up Transaction Processing	TBD *(to be defined)
Batch Run 2 (Cycle)	Interest Posting Minimum Payment	12/06/09
Batch Run 3 (Daily)	Interest Accrual	TBD *(to be defined)

Data Build

Identify plan to get data for each testing area. Mention the approach and activity plan for getting test data properly. The process step of prepare data mention here.

[Document the strategies that will be used to generate or extract appropriate data for the test effort].

Results/Sign-Off

Suspension/Resumption Criteria

Testing will halt for a particular project item (or function) when:

A critical problem is identified and where the potential code fix will require substantial re-testing of that function

It is identified that the business or technical specifications require major modifications due to escalated test issues and those modifications would require additional test analysis and or modification to the Detailed Test Plan.

The test regions or test environment are not available (for any reason).

The test regions or test environment suffer performance problems below 50% of their normal operating capacity, such that a region fix will require substantial re-testing of that function.

[Document all other suspension/resumption. Make sure the resumption criteria are unambiguously defined].

<Select from below table only testing level related on this DTP> consider the entry and Exit criteria reasonable on your project>

Pass/Fail Criteria

The specific pass/fail criteria for the testing at both the test cycle and release level are identifying in table below. This can include percentage of severity 3 and 4 defects that will be allowed to migrate between test/production environment and any specific business defined criteria. Identify information in this part , select only current testing level involve in this DTP.

[Document any specific pass/fail criteria for the testing at both the test cycle and the overall test activities. This can include percentage of severity 3 and 4 defects that will be allowed to migrate between test/production regions and any specific business defined criteria].

Table 7: Entry and Exit criteria of testing level

Testing Level	Entry criteria Guideline	Exit Criteria Guideline
UIS testing	Test environment available with latest software build Updated requirements documents (including change requests) Component designs	U/I/S testing objectives met All outstanding errors documented and assigned a severity level agreed with the Vendor manager All severity critical and high errors corrected or with agreed short-term

Testing Level	Entry criteria Guideline	Exit Criteria Guideline
	signed-off Interface designs (e.g. message formats/API protocols) agreed with Architecture team and with dev teams Environment configuration data has been defined and set-up	workarounds Full defect logs from final cycle of testing available for review Sample test plans/scripts available for review Test summary report distributed Software release packaged and under source control
SIT testing	System, Pre-SIT test exit criteria met Business requirements and specification documents signed-off Test environment available with latest software build deployed Environment configuration data has been defined and set-up Consolidated release note available SIT test preparation complete SIT risk based schedule agreed by all parties	SIT testing objectives met All test cases have been executed at least once (100% execution coverage) All outstanding errors documented and assigned a severity level agreed with the release management and vendor management. All severity critical and high errors corrected or with agreed short-term workarounds SIT testing analysis complete Test summary report distributed and approved
UAT testing	SIT exit criteria met Business requirements signed-off Business process maps complete and signed-off Training material available Test environment available with latest software build deployed	UAT testing objectives met All test cases have been executed at least once (100% execution coverage) All outstanding errors documented and assigned a severity level agreed with management team All severity critical and high errors

Testing Level	Entry criteria Guideline	Exit Criteria Guideline
	Environment configuration data has been defined and set-up	corrected or have documented workarounds formally agreed with business Test summary report distributed and approved Formal UAT Sign-off from K-Bank received

Test Conditions

Business Event 1- BR001

Testing will demonstrate the following:

Business Event 2 – BR002

Testing will demonstrate the following:

Business Event 14 – BR0014

Testing will demonstrate the following:

There is any financial transaction from dormant account. The dormant account means the account does not have any movement from customer action after end of the first due for 365 days. This excludes all batch jobs i.e. interest posting.

Product	Start Date	Due Date	Open A/C / Sub A/C	

3 Month	1/1/2013	31/3/2013	90	(1)
3 Month	1/4/2013	30/6/2013	90	(2)
3 Month	1/7/2013	30/9/2013	90	(3)
3 Month	1/10/2013	31/12/2013	90	(4)
3 Month	1/1/2014	31/3/2014	90	(5)
3 Month	1/4/2014	30/6/2014	90	(6)

The first due of 3-Month fixed account (1), 31/3/2013 + 365 days = 31/3/2014

If there is no customer transaction within 31/3/2014, this 3-Month fixed account is called dormant account.

Application Name: S1-ET, CT-Win

Test Environments

List and/or graphically show the proposed testing environment/s. For small testing projects complete all of the listed sections. Larger projects may require a separate Test Environment Plan to be produced. If Master test plan already specify may refer to MTP plan instead.

Client Side Infrastructure

Provide detailed list/s of required hardware and software at Client side to support testing activity of each test level and test environment. May use tabular format for explain the content.

Host/Server Side Infrastructure

Provide detailed list/s of required hardware and software at Host or Server to support testing activity of each test level and test environment. May use tabular format for explain the content.

Middleware

Provide detailed list/s of required Middleware to support testing activity of each test level and test environment. Eg. Test engine or test stub required for interface test.

Test Data preparation

Define the data subset/s that needs to be pre-loaded into the test environments. This content in is mention about infrastructure preparation view, it does not about the data condition or concept that mention in section 3.2 Data test.

[Provide detailed data requirements. Where specific data is required to establish a data condition or execute a test script provide either a list of these data requirements or reference the location of this information. Include the processes that will be used to deliver the data to the test region. (production extract by operations staff, build by automated scripts, generated by the development team/vendor etc)].

Test Schedule

[Provide a schedule showing how testing will be divided for execution. Use the following as an indicative sample. If a separate schedule is used provide an appropriate docref: and link and reference the rules governing its management].

Batch Run #	Function to be Executed	Cycle Date Req
Batch Run 1 (Daily)	Account Enquiries New Account Set-up Transaction Processing	NIL
Batch Run 2 (Cycle)	Interest Posting Minimum Payment	12/06/09
Batch Run 3 (Daily)	Interest Accrual	NIL

Appendix A - System Schematic/s

Clearly document the System Schematic used for test on this level.

13. ADITIONAL EXERSIZES

Foundation Level Exam 1 – Test your knowledge! (Answers are at and of test)

1 We split testing into distinct stages primarily because:

a) Each test stage has a different purpose.
b) It is easier to manage testing in stages.
c) We can run different tests in different environments.
d) The more stages we have, the better the testing.

2 Which of the following is likely to benefit most from the use of test tools providing test capture and replay facilities?

a) Regression testing
b) Integration testing
c) System testing
d) User acceptance testing

3 Which of the following requirements is testable?

a) The system shall be user friendly.
b) The safety-critical parts of the system shall contain 0 faults.
c) The response time shall be less than one second for the specified design load.
d) The system shall be built to be portable.

4 Analise the following highly simplified procedure:

Ask: “What type of ticket do you require, single or return?”
IF the customer wants ‘return’
Ask: “What rate, Standard or Cheap-day?”
IF the customer replies ‘Cheap-day’
Say: “That will be £11:20”
ELSE
Say: “That will be £19:50”
ENDIF
ELSE
Say: “That will be £9:75”
ENDIF

Now decide the minimum number of tests that are needed to ensure that all the questions have been asked, all combinations have occurred and all replies given.

a) 3
b) 4
c) 5
d) 6

5 Error guessing:

a) supplements formal test design techniques.
b) can only be used in component, integration and system testing.

c) is only performed in user acceptance testing.
d) is not repeatable and should not be used.

6 Which of the following is NOT true of test coverage criteria?
a) Test coverage criteria can be measured in terms of items exercised by a test suite.
b) A measure of test coverage criteria is the percentage of user requirements covered.
c) A measure of test coverage criteria is the percentage of faults found.
d) Test coverage criteria are often used when specifying test completion criteria.

7 In prioritizing what to test, the most important objective is to:
a) find as many faults as possible.
b) test high risk areas.
c) obtain good test coverage.
d) test whatever is easiest to test.

8 Which one of the following statements about system testing is NOT true?
a) System tests are often performed by independent teams.
b) Functional testing is used more than structural testing.
c) Faults found during system tests can be very expensive to fix.
d) End-users should be involved in system tests.

9 Which of the following is false?
a) Incidents should always be fixed.
b) An incident occurs when expected and actual results differ.
c) Incidents can be analyzed to assist in test process improvement.
d) An incident can be raised against documentation.

10 Enough testing has been performed when:
a) time runs out.
b) the required level of confidence has been achieved.
c) no more faults are found.
d) the users won't find any serious faults.

11 Which of the following is NOT true of incidents?
a) Incident resolution is the responsibility of the author of the software under test.
b) Incidents may be raised against user requirements.
c) Incidents require investigation and/or correction.
d) Incidents are raised when expected and actual results differ.

12 How would you estimate the amount of re-testing likely to be required?
a) Metrics from previous similar projects
b) Discussions with the development team
c) Time allocated for regression testing
d) a & b

13 Which of the following is true of the V-model?
a) It states that modules are tested against user requirements.
b) It only models the testing phase.
c) It specifies the test techniques to be used.
d) It includes the verification of designs.

14 Which of the following characterizes the cost of faults?

a) They are cheapest to find in the early development phases and the most expensive to fix in the latest test phases.

b) They are easiest to find during system testing but the most expensive to fix then.

c) Faults are cheapest to find in the early development phases but the most expensive to fix then.

d) Although faults are most expensive to find during early development phases, they are cheapest to fix then.

15 Which of the following should NOT normally be an objective for a test?

a) To find faults in the software.

b) To assess whether the software is ready for release.

c) To demonstrate that the software doesn't work.

d) To prove that the software is correct.

16 Which of the following is a form of functional testing?

a) Boundary value analysis

b) Usability testing

c) Performance testing

d) Security testing

17 Which of the following would NOT normally form part of a test plan?

a) Features to be tested

b) Incident reports

c) Risks

d) Schedule

18 Which of these activities provides the biggest potential cost saving from the use of CAST?

a) Test management

b) Test design

c) Test execution

d) Test planning

19 Which of the following is NOT a white box technique?

a) Statement testing

b) Path testing

c) Data flow testing

d) State transition testing

20 An important benefit of code inspections is that they:

a) enable the code to be tested before the execution environment is ready.

b) can be performed by the person who wrote the code.

c) can be performed by inexperienced staff.

d) are cheap to perform.

21 Which of the following is the best source of Expected Outcomes for User Acceptance Test scripts?

a) Actual results

b) Program specification
c) User requirements
d) System specification

22 What is the main difference between a walkthrough and an inspection?
a) An inspection is lead by the author, whilst a walkthrough is lead by a trained moderator.
b) An inspection has a trained leader, whilst a walkthrough has no leader.
c) Authors are not present during inspections, whilst they are during walkthroughs.
d) A walkthrough is lead by the author, whilst an inspection is lead by a trained moderator.

23 Which one of the following describes the major benefit of verification early in the life cycle?
a) It allows the identification of changes in user requirements.
b) It facilitates timely set up of the test environment.
c) It reduces defect multiplication.
d) It allows testers to become involved early in the project.

24 Integration testing in the small:
a) tests the individual components that have been developed.
b) tests interactions between modules or subsystems.
c) only uses components that form part of the live system.
d) tests interfaces to other systems.

25 Static analysis is best described as:
a) the analysis of batch programs.
b) the reviewing of test plans.
c) the analysis of program code.
d) the use of black box testing.

26 Alpha testing is:
a) post-release testing by end user representatives at the developer's site.
b) the first testing that is performed.
c) pre-release testing by end user representatives at the developer's site.
d) pre-release testing by end user representatives at their sites.

27 A failure is:
a) found in the software; the result of an error.
b) departure from specified behavior.
c) an incorrect step, process or data definition in a computer program.
d) a human action that produces an incorrect result.

28 The most important thing about early test design is that it:
a) makes test preparation easier.
b) means inspections are not required.
c) can prevent fault multiplication.
d) will find all faults.

29 Which of the following statements about reviews is true?
a) Reviews cannot be performed on user requirements specifications.
b) Reviews are the least effective way of testing code.

c) Reviews are unlikely to find faults in test plans.
d) Reviews should be performed on specifications, code, and test plans.

30 Test cases are designed during:
a) test recording.
b) test planning.
c) test configuration.
d) test specification.

31 A configuration management system would NOT normally provide:
a) linkage of customer requirements to version numbers.
b) facilities to compare test results with expected results.
c) the precise differences in versions of software component source code.
d) restricted access to the source code library.

Answers for above questions:

Question	Answer	Question	Answer
1	A	17	B
2	A	18	C
3	C	19	D
4	A	20	A
5	A	21	C
6	C	22	D
7	B	23	C
8	D	24	B
9	A	25	C
10	B	26	C
11	A	27	B
12	D	28	C
13	D	29	D
14	A	30	D
15	D	31	B
16	A		

EXERCISE 1 [GOOGLE HOMEPAGE TEST]

Find more than 20 defects (layout inconsistencies, spelling errors, and the like) in the image below:

The image is from Google Homepage (Long time ago). The current actual Google Homepage looks different from the image but the defects in the image are still valid and timeless. The image was edited using GIMP, and HTML using Notepad++.

And, yes, there are indeed more than twenty defects!

EXERCISE 2 [FAILED SHOT TEST]

A woman fired a shot at a man with her gun but the man did not die. List the possible reasons for the man not dying.

There are numerous possibilities above and this exercise helps a tester identify possible test scenarios/cases.

EXERCISE 3 [ROOM TEST]

List the defects/enhancements in the room you are in right now. [For example: there are dirty marks on the wall; the lighting could be better]

This exercise is to test and help develop your observance.

EXERCISE 4 [BRUSHING TEETH TEST]

An alien meets you and it asks you to teach it how to brush its teeth. Assume that the alien has teeth exactly like yours and is as smart as you but it needs a clear step-by-step instruction. List the steps. Be as detailed as you can. [Example: hold the toothpaste with your left hand; turn the cap anti-clockwise]

EXERCISE 5 [BALLPOINT PEN TEST]

Hold a ballpoint pen. Identify the types of testing you would perform on it to make sure that it is of the highest quality.

One can in fact associate almost all kinds of software testing types *while testing a pen.*

EXERCISE 6 [MOUSE TEST]

Similar to the Ballpoint Pen Test above, identify the types of testing you would perform on a mouse to make sure that it is of the highest quality.

Of course, we mean a computer mouse, and not the animal mouse, when we say 'mouse' here.

EXERCISE 7 [ADDITION TEST]

There is a simple program with the following items:

- Input Box A
- Input Box B
- ADD button
- Result Text Box [=A+B]

Identify all the test cases for the program. [Example: press the Add button without entering anything in Input Box A and B]

EXERCISE 8 FINDING DEFECTS

As a software tester, what do you do? Of course, testing the software, you would say……Okay, can you find out defects on the page shown below?

Here is how you can judge yourself:

If you find:

0 – 4 defects => Poor
5 – 6 defects => Average
7 – 8 defects => Good
9 – 10 defects => Excellent
10+ defects => Best tester!

(Mind well, you are the judge and you need to count on valid defects)

Here is one example of a defect as a hint:

– CONFIRM PASSWORD FIELD DOES NOT SHOW CONTENT IN ENCRYPTED MODE.

EXERCISE: FINDING DEFECTS ANSWERS

Defects :

1. *The user Id field accepts special characters.*
2. *Confirm password field does not show content in encrypted mode.*
3. *The name field does not seem to have any validation for number of characters.*
4. *Captcha is not at all readable.*
5. *There is no way user can reload the captcha.*
6. *Register button should be at bottom rather than on side.*
7. *Register button's label has "r" instead of "R".*
8. *There is no cancel button available if user wants to cancel the procedure..*
9. *There is no close button available if user wants to close the page.*
10. *The page title show wrong spelling of Registration.*

11. *Password selection guideline should be provided like the password should be alpha numeric or password strength factor should be present.*
12. *Page title should be New User Registration rather than New Registration.*
13. *Field length and labels should be same for the whole page / form.*
14. *The country field should by default show "Select" rather than selecting a value default.*

EXERCISE 9 WRITING TEST SCENARIOS

We do not only test, but we also develop test scenarios too. There is a real-life scenario and you need to write test ideas for the same.

Again, we do not want a step-by-step procedure; we want ideas with a brief description.

Write test ideas for this Scenario: You are at the grocery store's checkout counter. You have bought five items (x, y, z, a, and b). You make payment and move to the EXIT door.

Example Test ideas as a hint:

1. If the checkout counter is humanless, scan all the five items, scan your card and make payment.
2. The scanners should scan proper relevant information.

EXERCISE: WRITING TEST SCENARIOS ANSWERS

1. *If the checkout counter is human less, scan all the five items, scan your card and make payment.*
2. *The scanners should scan proper relevant information.*
3. *If the checkout counter is human processed, a person to help should be available.*
4. *All the items bought should have barcode so that they are scan able.*
5. *All the items should have MRP printed and scan able so that the software can read it*
6. *The relevant software and printers should be in working condition*
7. *Once all items scanned, a bill should be generated and given to the customer.*
8. *For payment multiple options should be allowed, cash, card (credit card, debit card), coupons (meal passes) etc.*
9. *If payment is done by card, the transactions should be secured.*
10. *If payment is done by cash, counter person must have enough cash to balance the bill.*
11. *If any of the item bought is not scannable, the counter person should be able to help customer.*
12. *Customer should be able to see the EXIT sign easily*
13. *At EXIT, there should be some check that customer carries only the bought and billed items.*

EXERCISE 10 DEFECT REPORTING

As a tester, the best part of the job is to report defects. We would like to know how you would report the following defect (you can decide upon the fields you want to include while reporting the defect in the best way).

Write a detailed defect report for this sample defect: After logging into Gmail, it navigates to Google.com
No hint here. :) Just write a good and complete defect report.

EXERCISE: DEFECT REPORTING ANSWERS

Defect: After logging into Gmail, it navigates to Google.com
Title – Gmail login navigates to Google home page rather than mail inbox.
Severity : High
Priority : Critical
Observed on : Windows – FF x.x and IE x.x and Mac – Chrome x
Module : Login

Reproduction Steps :
1. Launch a Gmail link with compatible browsers.
2. Click on Sign in button in link, Gmail login window screen is appeared.
3. Enter a valid login credentials.
4. Click on SIGN IN button,

Analysis :
Defect Status : Open
Assigned to : Project Manager x.x

EXERCISE 11 PROVIDING SUGGESTIONS

Providing suggestions to improve quality or user experience is the extension of a Software Testing job. So why not try that? Can you tell us how user experience can be improved for the following sign-in page?

Here is an example suggestion as a hint:

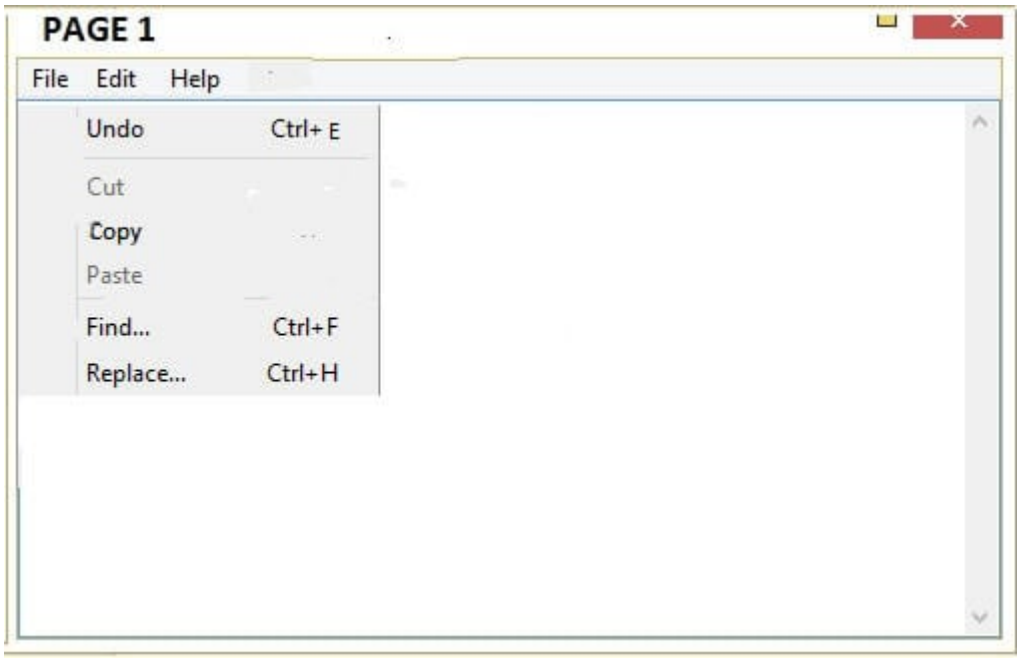
– Rather than asking the customer to select whether he is a new customer, the system should check the status of the customer based on the e-mail address or user id he had provided.

EXERCISE: PROVIDING SUGGESTIONS ANSWERS

1. *Rather than the question "What is your e-mail address?", it should be simply "e-mail address".*
2. *Sign in facility should be available via e-mail as well as userID.*
3. *Rather than asking customer to select whether he is new customer, system should check status of customer based on e-mail address or userID he had provided.*
4. *Most of the time, end user does not concern about info that system is using secure server.it is assumed by default. So the button should be simply Sign In.*
5. *Help link should be available for any customer who is stuck due to any reason.*
6. *Page title should be backgrounded.*
7. *A close button should be available.*
8. *A Cancel button should be available.*

EXERCISE 12 BUG HUNTING

One of the trainee developers has developed a **text pad kind of application**. Below is one of the screenshots of the application. **Can you list out bugs/issues, which the trainee developer should take care of?**



Sample Answer as a hint:
The name of the application does not appear in Title space.

EXERCISE: BUG HUNTING ANSWERS

1. The name of application does not appear on Title space.
2. The title space seems cutting on right side where close button is placed.
3. No button for minimizing.
4. The Edit menu should be displayed in a ways that the menu's left wall should be aligned to Edit option.
5. Copy option is enabled by default.
6. For Undo the generalized shortcut key is Ctrl+Z and its difficult for users to get used to with different keys for the default action like Undo.
7. For Cut, Copy and Paste, no shortcut keys have been displayed / provided.
8. The Title bar does not show application logo.
9. The Title should be centralist.
10. The Edit menu seems to be incomplete at end.

EXERCISE 13 THINK TESTING

Food for the brain – you need to think now, really. :)

We are living in the Eco Era, right? Our question is also related to Green Product –

HOW WOULD YOU TEST A 5 KG CAPACITY GROCERY SHOPPING PAPER BAG?
VERY SIMPLE? START WRITING UP YOUR TEST IDEAS IN TERMS OF *Test Scenarios*.

Sample Answer as a hint:
I would put 5KG package rice in it and will try to accommodate a 150 gr biscuit packet too as I do not have another bag to put the biscuit packet. Will like to see whether the paper bag (whose capacity is 5 kg) able to carry just extra weight till I travel for 10 minutes?

EXERCISE: THINK TESTING ANSWERS

1. I would put 5KG package rice in it and will try to accommodate a 150 gms biscuit packet too as I do not have another bag to put the biscuit packet. Will like to see whether the paper bag (whose capacity is 5 kgs) able to carry just extra weight till I travel for 10 mins?
2. Put exactly 5 KG wheat flour packet in it and carry it for half an hour while walking around.
3. Put 10 items which sums up as 5 kg in terms of weight.
4. Test scenarios for checking quality of paper used for the bag.
5. Test scenarios to check quality of handles of the bag.
6. Put 5 KG package of rice in the bag and just throw it in car dickey. Is the bag able to tolerate that punch?
7. Test scenarios of bag being kids friendly (what if child starts chewing it if he finds it on the floor)
8. Test scenarios to check overall look of bag.
9. Try to put 8 kG package of floor in it, how does it react?
10. Is the bag paper writable? I mean, if someone wants he should be able to label it.
11. What if the paper bag gets wet?
12. What if you try to accommodate it in the purse or another bag, by folding it?

EXERCISE 14 SHOW DEFECT REPORTING SKILLS

Defect reporting for the following issue:
Requirement: After registering to the site – example.com, a new user receives an e-mail, which contains a link to reset the default set password.
Issue: When the user registers via mobile, he receives the e-mail two times.
Log a defect report for this issue with all required defect report fields.

……
……
……
……
……
……
……
……
……
……
……
……
……
……
……
……

EXERCISE 15 ROLE OF AN END-USER

As a tester, we do not test the application product. We do play the role of an end-user. We do try to generate real-time scenarios where the end-user might not behave the way they are expected. How the product would respond, that is what we test.

Let's reverse the process to a small degree –

AS AN END-USER, HOW WOULD YOU LIKE TO MODIFY THE ATM (AUTOMATED TELLER MACHINE) SO THAT IT CAN BE MORE BENEFICIAL?

DON'T PROVIDE JUST IDEAS, PROVIDE RELEVANT REASONS TOO.

Sample Answer as a hint:

I would suggest removing "the cash withdrawal default setting" at the end of the transaction as most of the time, we do not want to set the same cash every time and so we

do not like to set it. It unnecessarily wastes time where you have to push NO and will have to wait for the debit card.

EXERCISE: ROLE OF AN END USER ANSWERS

1. I would suggest to remove “the cash withdrawal default setting” at the end of transaction as most of the time, we do not want to set the same cash every time and so we do not like to set it. It unnecessarily wastes time where you have to push NO and will have to wait for the debit card.
2. I would suggest to include face/finger based authorization rather than password/pin because password/pin can be hacked but face cannot be :)
3. The ATM machine should have card scanner rather than card insertion facility as sometimes the faulty machines does not work properly and card gets stuck.
4. The ATM machine should not provide facility for printed statement as it’s a waste of paper. If a person really needs it, he can manage from bank or via online banking. Printed statement is a waste of paper as 95% of user throws it immediately. We need to be more environmental conscious and should see the balance or statement online rather than taking print of every transaction.

TESTING CHALLENGE #1 - WEB TESTING

What to do

Identify at least 10 (max 18) tests required for the scenario below, based on the data you input in the First Name field.

What not to test for

Different browsers, extremely big requests, "nasty words", browser zoom in and out. Do not use automation tools. The server will cut access at over 30 requests per second per IP.

Specification

The user has to fill in the required data in order to get access, as a standard user in a forum.

Only the First Name field can be tested right now. The field has a max length of 30.

Confirmation message

Thank you, Gheorghe, …… from **Norway**.

A message will be sent shortly to your e-mail address: **me@whatyouknow.com** with your password.

Your username is **Gheorghe** .

Country Norw ay **Email** me@w haty

First Name* **Last Name** Gheorghe

- Hint: Test for input Other chars then alphabetic

TESTING CHALLENGE #1 - WEB TESTING - ANSWERS

*** 10 possible checks:

- Space values at the beginning
- Space values at the end
- Non ASCII
- Maximum values
- Space
- Space in the middle
- More than maximum values
- Minimum value
- Other chars then alphabetic
- Average value

TESTING CHALLENGE #2 - GENERATE TESTING DATA

What to do

Create 5 valid data input for the specification below.
This challenge emphasizes the skill required to create your own test data.

Specification

Each person born in Romania receives a unique identification number.
The number has 13 digits, e.g. **1234567890123**

What the digits represent:

- First digit – the gender: male or female
  - 1 or 2 – born between 1 January 1900 and 31 December 1999
  - 3 or 4 - born between 1 January 1800 and 31 December 1899
  - 5 or 6 - born between 1 January 2000 and 31 December 2099
  - 7 or 8 – Foreign residents in Romania.
  - 9 - For non-residents
- Next 2 digits – last 2 digits of the year of birth (e.g. born in 1980 then it will 80)
- Next 2 digits – month of birth (01 to 12)
- Next 2 digits - date of birth (01 to 31 depending on the month of birth)
- Next 2 digits – area code (valid codes are 01 to 52)
- Next 3 digits – order number
- Last digit control number is created by:

Take the first 12 numbers of the CNP: **123456789012** and multiply each number with the corresponding position number from this
string: **279146358279** like: **1** * **2** + **2** * **7** + **3** * **9** + **4** * **1**+…..+ **12** * **9**= X

You divide X by 11 the rest obtained if it is 10 then the last digit is 1 otherwise it equals the rest obtained

List 5 correct CNP numbers: >>>>>>>>>>>

1) ……

2) ……

3) ……

4) ……

5) ……

TESTING CHALLENGE #3 - SCENARIO TESTING

Why this challenge?

As testers we have to come up with scenarios. We have to imagine them based on the information we have.
Scenarios are like stories. This challenge helps you practice your story telling.

Your challenge is to imagine and write a story, that has meaning, with the words bellow. You have 20 minutes to do so.

Your words are:

Developer Compatibility Alpha Beta Testing Software Crafsmanship

---

What scenario can you think of with the given words? Write below!

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……