ADAMA SCIENCE AND TECHNOLOGY UNIVERSITY

SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTING(SOEEC)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING(CSE)

SOFTWARE QUALITY AND TESTING (CSE5310)

Assignment 2

SECTION 3

GROUP MEMBERS

NATNAEL AFEWORK	A/UR14752/10
NATHANAEL BOGALE	A/UR14674/10
ELSABETH FENKESA	A/UR14727/10

SUBMITTED TO: Dr. T. Gopi Krishna

DATE: Apr 20, 2022

1. Write major differences between Regression and Re-testing approach techniques.

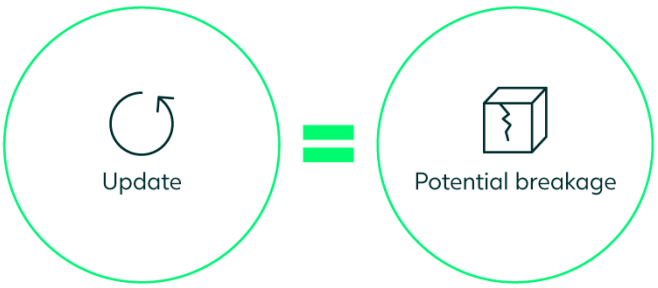
Regression Testing

Regression testing is a type of black-box testing. It ensures that any modification or addition to the existing code base has not adversely impacted the **previously developed and tested features**.

Regression testing confirms if the application is working as desired and expected after the change. Smoke and Sanity testing are often considered types of regression testing.

Both functional and non-functional tests are executed while performing regression testing. Regression testing is never performed on a new module

The objective of regression testing is to check that the new code changes do not negatively impact the existing developed and tested functionalities of the application.



To perform regression testing you typically have a regression suite – a series of test cases set up to test these older features.

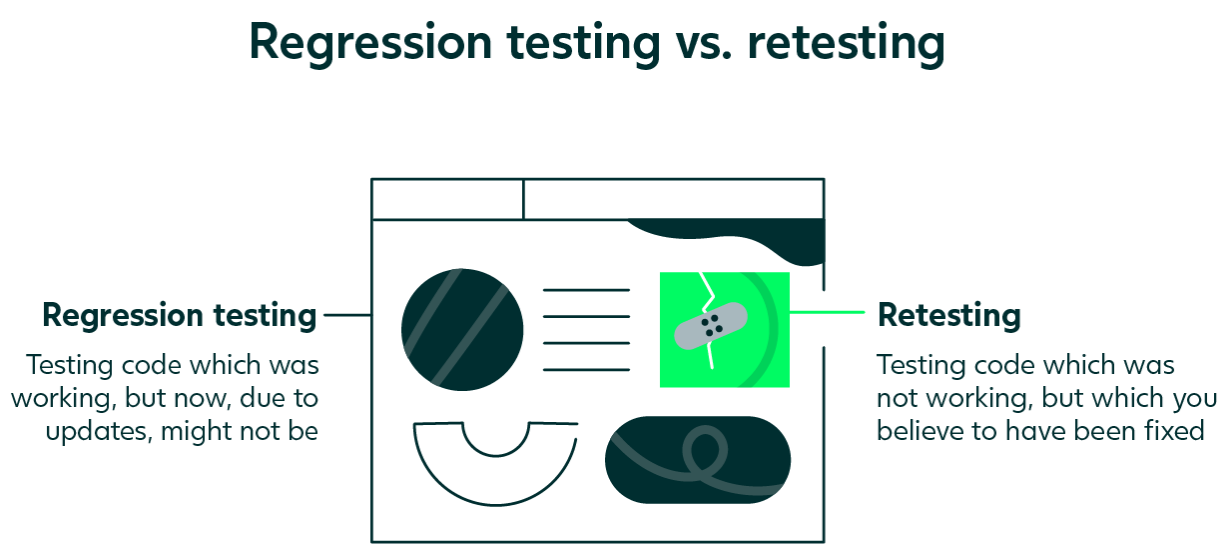
Regression test cases are often automated because these tests build up as the software changes or grow.

Retesting

Retesting is also known as confirmation testing. Retesting is to ensure that the defects raised during the SDLC (software development life cycle) are fixed and work according to the specifications. During retesting, the failed test cases are re-executed and passed.

The objective of retesting is to check **if the identified defects are fixed**. To verify the fix, the test cases linked to the defect are re-executed and passed.

Regression testing vs. retesting: key differences



We could say that regression testing is a type of retesting. Retesting essentially means to test something again. And when we are regression testing, we're testing something that we've tested numerous times before.

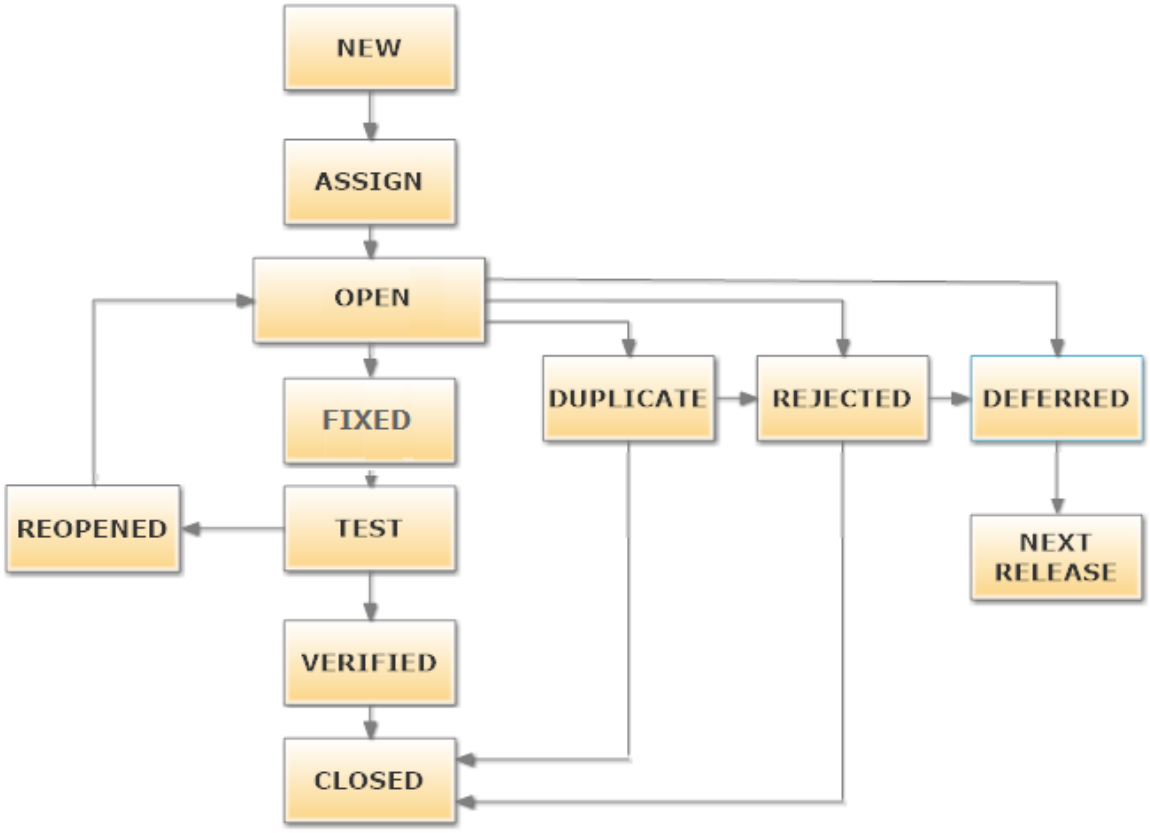
Regression Testing	**Retesting**
Involves testing a *general area* of the software.	Involves testing a *specific feature* of the software.
Is about testing software which *was* working, but now, due to updates, *might not* be working.	Is about testing software which you *know was not* working, but which you believe to have been fixed. You test it to confirm that it is now in fact fixed.
Is ideal for automation as the testing suite will grow with time as the software evolves.	*Is not ideal* for automation as the case for testing changes each time.
Should *always* be a part of the testing process and performed each time code is changed and a software update is about to be released.	Is *only* a part of the testing process if a defect or bug is found in the code.

2. Explain the bug life cycle stages with a neat diagram and describe how bug priority can be differentiated from the severity of the bug.

The bug life cycle is also known as the Defect life cycle. In the Software Development Process, the bug has a life cycle. The bug should go through the life cycle to be closed.

- Once the bug is identified by the tester, it is assigned to the development manager in open status
- If the bug is a valid defect the development team will fix it and if it is not a valid defect, the defect will be ignored and marked as rejected
- The next step will be to check whether it is in scope, if it is happen so that, the bug is not the part of the current release then the defects are postponed

- If the defect or bug is raised earlier then the tester will assigned a DUPLICATE status
- When bug is assigned to developer to fix, it will be given a IN-PROGRESS status
- Once the defect is repaired, the status will change to FIXED at the end the tester will give CLOSED status if it passes the final test.



Defect Life Cycle States

The different states of a bug in the software bug life cycle (sblc) are as follows:

#1. 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 the tester is “New”.

#2. Assigned

Defects that 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”

#3. Open

The development team starts analyzing and working on defect fix

#4. 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.

#5. Test

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

#6. 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.”

#7. Closed

After verifying the fix, if the bug no longer exists then the status of the bug will be assigned as “Closed.”

#8. Reopen

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

#9. Duplicate

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

#10. Deferred

In some cases, the Project Manager/Lead may set the bug status as deferred.

- If the bug is found during the end of the release and the bug is minor or not important to fix immediately.
- If the bug is not related to the current build.
- If it is expected to get fixed in the next release.
- The customer is thinking about changing the requirement.
- In such cases the status will be changed as “deferred” and it will be fixed in the next release.

#11. Rejected

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

Key Difference of Bug Priority and Severity of a Bug

- Priority is the order in which the developer should resolve a defect whereas Severity is the degree of impact that a defect has on the operation of the product.
- Priority is categorized into three types : low, medium and high whereas Severity is categorized into five types : critical. major, moderate, minor and cosmetic.

- Priority is associated with scheduling while Severity is associated with functionality or standards.
- Priority indicates how soon the bug should be fixed whereas Severity indicates the seriousness of the defect on the product functionality.
- Priority of defects is decided in consultation with the manager/client while Severity levels of the defects are determined by the QA engineer.
- Priority is driven by business value while Severity is driven by functionality.
- Priority value is subjective and can change over a period of time depending on the change in the project situation whereas Severity value is objective and less likely to change.
- High Priority and low severity status indicates, defects have to be fixed on immediate basis but does not affect the application while High Severity and low priority status indicates defects have to be fixed but not on immediate basis.
- Priority status is based on customer requirements whereas Severity status is based on the technical aspect of the product.

3. During the testing of a module tester “ABC” finds a bug and assigns it to the developer. But the developer rejects the same, saying that it is not a bug. What tester, “ABC”, should do?

Send to the detailed information of the bug encountered and check the reproducibility.

4. Describe positive testing and negative testing approaches with suitable examples.

Positive and Negative testing has prime importance when it comes to testing an app or software. It is said that testers should not only test for the positive results i.e. whether the system functions as expected with valid inputs but also check for negative value tests i.e. if the application works with quality under provided invalid inputs.

Positive Testing

Positive testing is a method of checking if the application does what it is expected to do. When a software tester writes the test cases for a set of specified outputs, it is called a positive test.

It is used to check whether our application works as expected or not. And if an error is detected at the time of positive testing, the test is considered as fail. A positive testing is a technique whenever a test engineer writes the test cases for a set of respective outputs.

In positive testing, the test engineer will always check for only a good set of values.

In other words, we can say that **positive testing** is a process where the system or an application is tested against the valid input data.

And the primary purpose of performing the positive testing is to validate whether the software does what it is supposed to do.

Example of Positive Testing

In the following example, we are trying to understand the working of positive testing.

- Suppose, we have one test scenario, where we want to test an application that includes a simple text box.
- To enter a Phone Number according to the business needs it accepts only numerical values.
- Hence, in positive testing, we will only give the positive numerical values in order to test whether it is working as per the requirement or not.

Negative Testing

It is implemented to check how the application can gracefully handle invalid input or unpredicted user performance.

The fundamental purpose of executing the negative testing is to ensure the application's stability against the effects of different variations of improper validation data set.

Negative testing is also known as **error path testing or failure**. And it helps us to identify more bugs and enhance the quality of the software application under test.

Once the positive testing is complete, then only we can execute the negative testing, which helps to identify more bugs and enhance the quality of the software application under test.

Example of Negative Testing

Let's see one example which will help us to understand the negative testing in an effective way.

- Suppose we have one sample form where the **Mobile Number Field** can only accept the numbers' value and does not accept the unique characters and alphabets values.
- However, let's enter the unique characters and alphabets values on **the Mobile number field** to test whether it accepts the individual characters and alphabets values or not.
- We expect that the textbox field will not accept invalid values and show an error message for the incorrect entry.

5. Explain the significance of Grey box testing, its advantages and disadvantages.

Gray box testing (a.k.a grey box testing) is a method you can use to debug software and evaluate vulnerabilities. In this method, the tester has limited knowledge of the workings of the component being tested. This is in contrast to black box testing, where the tester has no internal knowledge, and white box testing, where the tester has full internal knowledge.

You can implement gray box testing as a form of penetration testing that is unbiased and non-obtrusive. In these tests, the tester typically knows what the internal components of an application are but not how those components interact. This ensures that testing reflects the experiences of potential attackers and users.

Gray box testing is most effective for evaluating web applications, integration testing, distributed environments, business domain testing, and performing security assessments. When performing this testing, you should create clear distinctions between testers and developers to ensure test results aren't biased by internal knowledge.

Advantages of Grey Box Testing

- As grey box testing is a combination of black box and white box testing, it provides the best of both worlds i.e. benefits of both the testing techniques.
- Knowledge of the internal mechanisms of the system helps the tester to design test scenarios more extensively.
- For grey box testing, functional specifications and other design documents are used. It does not need the use of the source code which helps in keeping the source code safe from any disruptive changes.

- It helps in keeping testers and developers separate, which reduces any disagreement between them.
- Even with a partial understanding of the code, testers conduct grey box testing from the end user's perspective. This helps in identifying any issues that the developers might have missed during unit testing.
- It results in the instant fixing of the issues as a tester can change the partially available code to check for the results.
- Even without high-level programming skills, the testers can perform this testing.
- It is platform and language-independent.

Disadvantages of Grey Box Testing

- While doing grey box testing, testers do not have access to the source code, so it becomes difficult to get complete code path coverage and testers might fail to notice some critical vulnerabilities.
- Algorithm testing is not possible as accessing the complete logic of the algorithms is not possible.
- If a developer has already executed a test case, running the same test case in grey box testing may result in redundancy.
- It is usually not suitable for distributed systems.

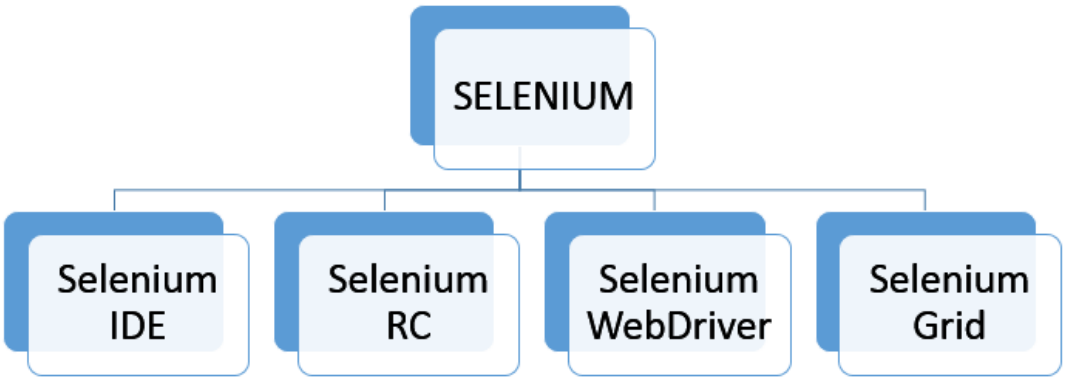
6. Describe major elements and the functionality of the selenium framework.

Selenium is an automation testing tool used to test web-based applications. Selenium is not a single tool but a suite of tools. There are four components of Selenium – Selenium IDE, RC, WebDriver, and Grid. Last two are the most famous one. Different components of Selenium provide different features – multiple browsers support, parallel test capabilities, execution on multiple machines and much more.

It can automate all websites and does not depend on the technology in which the application is designed. Performance and execution speed of Selenium Automation tool is much better than any other tools which are available in the market.

Selenium Components:

1. Selenium IDE (Integrated Development Environment).
2. Selenium RC(Remote Control)
3. Selenium WebDriver
4. Selenium Grid



Selenium IDE

Selenium IDE (Integrated Development Environment) is the major tool in the Selenium Suite. It is a complete integrated development environment (IDE) for Selenium tests. It is implemented as a Firefox Add-On and as a Chrome Extension. It allows for recording, editing and debugging of functional tests. It was previously known as Selenium Recorder.

Scripts may be automatically recorded and edited manually providing autocompletion support and the ability to move commands around quickly. Scripts are recorded in Selenese, a special test scripting language for Selenium. Selenese provides commands for performing actions in a browser (click a link, select an option) and for retrieving data from the resulting pages.

Selenium RC (Remote control)

Selenium Remote Control (RC) is a server, written in Java, that accepts commands for the browser via HTTP. RC makes it possible to write automated tests for a web application in any programming language, which allows for better integration of Selenium in existing unit test frameworks. To make writing tests easier, Selenium project currently provides client drivers for PHP, Python, Ruby, .NET, Perl and Java. The Java driver can also be used with JavaScript (via the Rhino engine). An instance of selenium RC server is needed to launch html test case – which means that the port should be different for each parallel run. However, for Java/PHP test case only one Selenium RC instance needs to be running continuously.

Selenium Web Driver

Selenium WebDriver is the successor to Selenium RC. Selenium WebDriver accepts commands (sent in Selenese, or via a Client API) and sends them to a browser. This is implemented through a browser-specific browser driver, which sends commands to a browser and retrieves results. Most browser drivers actually launch and access a browser application (such as Firefox, Google Chrome, Internet Explorer, Safari, or Microsoft Edge); there is also an HtmlUnit browser driver, which simulates a browser using the headless browser HtmlUnit.

Selenium WebDriver does not need a special server to execute tests. Instead, the WebDriver directly starts a browser instance and controls it. However, Selenium Grid

can be used with WebDriver to execute tests on remote systems (see below). Where possible, WebDriver uses native operating system level functionality rather than browser-based JavaScript commands to drive the browser. This bypasses problems with subtle differences between native and JavaScript commands, including security restrictions.

Selenium GRID

Selenium Grid is a server that allows tests to use web browser instances running on remote machines. With Selenium Grid, one server acts as the hub. Tests contact the hub to obtain access to browser instances. The hub has a list of servers that provide access to browser instances (WebDriver nodes), and lets tests use these instances. Selenium Grid allows running tests in parallel on multiple machines and to manage different browser versions and browser configurations centrally (instead of in each individual test).

The ability to run tests on remote browser instances is useful to spread the load of testing across several machines and to run tests in browsers running on different platforms or operating systems. The latter is particularly useful in cases where not all browsers to be used for testing can run on the same platform.

7. Write Test Cases using Boundary Value Analysis for a requirement that is stated as follows:

“In the examination grading system, if the student scores 0 to less than 40 then assign E Grade, if the student scores between 40 to 49 then assign D Grade, if the student scores between 50 to 69 then assign C Grade, if the student scores between 70 to 84 then assign B Grade, and if the student scores 85 to 100 then assign A Grade.”

0 to 39 -E
40 to 49-D
50 to 69-C
70 to 84-B
85 to 100-A
Based on BVA, we identify following input values to test each boundary: For Boundary Values in range 0 to 39:
For 0 to 39 boundary values, Minimum Value= 0, Maximum Value = 39, Precision is 1.
Thus, input values for testing for this boundary values are: (Minimum Value-precision)= -1 Minimum Value= 0
(Minimum Value+ precision)= 1 (Maximum Value- precision)= 38 Maximum Value= 39 (Maximum Value + precision)= 40
Thus, input values for Boundary Values 0 and 39 are: -1, 0, 1, 38, 39, 40.
On the similar lines of analysis, we arrive at following input values for other Boundary values:
For 40 to 49, we have 39, 40, 41, 48, 49, 50
For 50 to 69, we have 49, 50, 51, 58, 59, 60
For 70 to 84, we have 69, 70, 71, 83, 84, 85 For 85 to 100, we have 84, 85, 86, 99, 100, 101

8. If you hold an 'over60s' railcard, you get a 34% discount on whatever ticket you buy. If you are traveling with a child (under 16), you can get a 50% discount on any ticket if you hold a family railcard, otherwise, you get a 10% discount. You can only hold one type of railcard. Construct a decision table showing all the combinations of fare types and resulting discounts and

derive test cases from the decision table.

Solution: There are 3 conditions, so there can be 8 combinations of rules.

Causes (input)	R1	R2	R3	R4	R5	R6	R7	R8
Over 60s railcard?	Y	Y	Y	Y	N	N	N	N
Family railcard?	Y	Y	N	N	Y	Y	N	N
child also traveling?	Y	N	Y	N	Y	N	Y	N
Effects (Output)								
Discount (%)	X/?/50 %	Y/?/34 %	34 %	34 %	50 %	0 %	10 %	0 %

Question mark is added because specification does not say what happen if somebody holds more than 2 cards. If someone holds 2 cards probably he would not admit this, and can claim a discount. Check Rules 3 and 4, both output 34%, so 3rd cause does not matter.

Causes (input)	R1	R2	R3	R5	R6	R7
Over 60s railcard?	Y	Y	Y	N	N	N

Family railcard?	Y	Y	N	Y	-	N
child also traveling?	Y	N	-	Y	N	Y
Effect (Output)						
Discount (%)	50%	34%	34%	50%	0%	10%

Here table can be rationalized like this to reduce test cases. R6 and R8 also combined because having a family railcard has no effect if you are not traveling with a child.

Here are the test cases

TCID	**Input**	**Outcome**
1	X is traveling with 2 card Over 60s and Family along with his grandson	50% discount for both tickets
2	X is traveling alone with 2 card Over 60s and Family Card	34% discount
3	X traveling with his wife along with Over 60s card	34% discount
4	X traveling with family card and grandson	50% discount

5	X traveling alone without any card	0%
6	X traveling with grandson without any card	10% discount for both tickets

9.**a. Explain differences between manual testing and automation testing.**

Manual testing and automated testing cover two vast areas. Within each category, specific testing methods are available, such as black box testing, white box testing, integration testing, system testing, performance testing, and load testing. Some of these methods are better suited to manual testing, and some are best performed through automation. Here's a brief comparison of each type, along with some pros and cons:

Manual Testing	Automated Testing
● Manual testing is not accurate at all times due to human error, hence it is less reliable.	● Automated testing is more reliable, as it is performed by tools and/or scripts.
● Manual testing is time-consuming, taking up human resources.	● Automated testing is executed by software tools, so it is significantly faster than a manual approach.
● Investment is required for human resources.	● Investment is required for testing tools.

• Manual testing is only practical when the test cases are run once or twice, and frequent repetition is not required.	• Automated testing is a practical option when the test cases are run repeatedly over a long time period.
• Manual testing allows for human observation, which may be more useful if the goal is user-friendliness or improved customer experience.	• Automated testing does not entail human observation and cannot guarantee user-friendliness or positive customer experience.

b. In which phase of the product lifecycle manual testing is preferable?

manual testing is a process in which we compare the behavior of a piece of software (it can be a component, module, feature, etc.) with the predefined, expected behavior which we set during the initial phases of SDLC.
Manual verification is the most primitive form of software testing. A novice can do it without any knowledge of any particular tool. Even a student, who has a basic understanding of the application or testing of a system, can perform manual verification. Nonetheless, it is an essential step in the software testing cycle. Any new system or applications must be tested manually before automating the testing.

c. Write the advantages of automation testing.

Saves time

Automating the testing process helps the testing team to use less time to validate newly created features. For instance, in manual testing, there is a need to write thousand test cases for a calculator application, but automation makes the process much faster.

Productivity improvement

As during execution, automation tests do not require human intervention, so testing an application can be done late at night, and we can get the results next morning. Software developers and testers require less time on automation testing.

Accuracy improvement

In manual testing, there is a chance of mistakes whether you are an experienced testing engineer. The chances of errors may increase when testing a complex use case. But Automation testing reduces the chances of errors. There is good accuracy, as we will get the same result each time on performing the same test cases.

Test suite reusability

We can reuse the test scripts in automation testing, and we don't need to write the new test scripts again and again. These test cases can be used in various ways, as they are reusable. Reusability helps to reduce the cost and also eliminate the chances of human error.

Ability to test on various platforms

Automation testing allows the user to test the application on different web browsers and operating systems.

Running tests 24/7

In automation testing, we can start the testing process from anywhere in the world and anytime we want. It can also be done remotely if we don't have many approaches or the option to purchase them.

Early bug detection

By automation testing, it is easy to detect critical bugs in the initial phases of software development. It reduces the cost and helps us to spend fewer working hours to fix such problems. It increases the efficiency of the team.

Less human resources

Automation testing requires fewer people to perform a tedious manual test. To implement the automation test script, we need a test automation engineer who can write the test scripts to automate our tests.

Reduce the expenses

Automation testing is less expensive, as once the test scripts have been built, we can reuse them at any time without any extra cost. While manual testing is more expensive than automation, with manual monitoring, it is typical to execute experiments repeatedly.

Scalability of test cases

In manual testing, we require the involvement of the number of people and number of hours to scale up a project. Whereas the scalability of automation testing is higher, we need adding of test executors to the testing framework.

Consistency

Compared to manual testing, automation testing is more consistent and way faster than executing the regular monotonous tests that cannot be missed but may cause faults when tested manually.

Fast development and delivery

Automated tests can be executed repeatedly and completed rapidly. We do not have to wait for weeks to execute the tests; few hours are enough for execution. Switching from manual to automation reduces the waiting time and boosts development.

Easily execution of lengthy and complicated test cases

Execution of bug-prone and complex test cases is easier with automation testing. Test cases with reproducible steps lead to distraction and wrong assurances on testing them manually.

Some of the other benefits of automation testing are listed as follows -

- In comparison to manual testing, automation testing requires fewer resources.
- It makes load and performance testing, stress testing, and reliability testing possible.
- It is more reliable, as it reduces the occurrence of errors. It is reliable because it tests the application with the help of tools and test scripts.
- With automation testing, test engineers are free to focus on other work.
- It improves the testing coverage as the automatic execution of test cases is faster than manual execution.
- Automation testing allows the execution of test cases in a 24x7 environment.
- It enhances the knowledge of test engineers by producing a repository of different test cases.
- Batch execution is possible using automation testing because all the written scripts can be executed simultaneously.
- Automation testing is 70% faster than manual testing.

d. Explain differences between smoke testing and monkey testing Techniques.

- Monkey testing is random testing, and smoke testing is a nonrandom testing. Smoke testing is nonrandom testing that deliberately exercises the entire system from end to end, with the the goal of exposing any major problems.

- Monkey testing is performed by automated testing tools, while smoke testing is usually performed manually.
- Monkey testing is performed by "monkeys", while smoke testing is performed by skilled testers.
- "Smart monkeys" are valuable for load and stress testing, but not very valuable for smoke testing, because they are too expensive for smoke testing.
-"Dumb monkeys" are inexpensive to develop, are able to do some basic testing, but, if we used them for smoke testing, they would find few bugs during smoke testing.
- Monkey testing is not a thorough testing, but smoke testing is thorough enough that, if the build passes, one can assume that the program is stable enough to be tested more thoroughly.
-Monkey testing either does not evolve, or evolves very slowly. Smoke testing, on the other hand, evolves as the system evolves from something simple to something more thorough.
- Monkey testing takes "six monkeys" and a "million years" to run. Smoke testing, on the other hand, takes much less time to run, i.e. from a few seconds to a couple of hours.

10.Explain following testing techniques

a. Usability testing

Usability testing refers to evaluating a product or service by testing it with representative users. Typically, during a test, participants will try to complete typical tasks while observers watch, listen and takes notes. The goal is to identify any usability problems, collect qualitative and quantitative data and determine the participant's satisfaction with the product

b. Performance testing

Performance testing is a non-functional software testing technique that determines how the stability, speed, scalability, and responsiveness of an application holds up under a given workload. It’s a key step in ensuring software quality, but unfortunately, is often seen as an afterthought, in isolation, and to begin once functional testing is completed, and in most cases, after the code is ready to release.

The goals of performance testing include evaluating application output, processing speed, data transfer velocity, network bandwidth usage, maximum concurrent users, memory utilization, workload efficiency, and command response times.

c. Stress testing

Stress testing is the process of determining the ability of a computer, network, program or device to maintain a certain level of effectiveness under unfavorable conditions. The process can involve quantitative tests done in a lab, such as measuring the frequency of errors or system crashes. The term also refers to qualitative evaluation of factors such as availability or resistance to denial-of-service (DoS) attacks. Stress testing is often done in conjunction with the more general process of performance testing.

When conducting a stress test, an adverse environment is deliberately created and maintained. Actions involved may include:

- Running several resource-intensive applications in a single computer at the same time
- Attempting to hack into a computer and use it as a zombie to spread spam
- Flooding a server with useless E-mail messages
- Making numerous, concurrent attempts to access a single Web site
- Attempting to infect a system with viruses, Trojans, spyware or other malware.

d. Scalability testing

Scalability Testing is a type of non-functional testing in which the performance of a software application, system, network or process is tested in terms of its capability to scale up or scale down the number of user request load or other such performance attributes. It can be carried out at a hardware, software or database level. Scalability Testing is defined as the ability of a network, system, application, product or a process to perform the function correctly when changes are made in the size or volume of the system to meet a growing need. It ensures that a software product can manage the scheduled increase in user traffic, data volume, transaction counts frequency and many other things. It tests the system, processes or databases ability to meet a growing need.

e. Compatibility testing

A compatibility test is an assessment used to ensure a software application is properly working across different browsers, databases, operating systems (OS), mobile devices, networks and hardware. Compatibility testing is a form of non-functional software

testing -- meaning it tests aspects such as usability, reliability and performance -- that is used to ensure trustworthy applications and customer satisfaction.
Compatibility tests are crucial to the successful performance of applications. They should be performed whenever a build becomes stable enough to undergo testing.

f. API Testing

API testing is a type of software testing that analyzes an application program interface (API) to verify it fulfills its expected functionality, security, performance and reliability. The tests are performed either directly on the API or as part of integration testing. An API is middleware code that enables two software programs to communicate with each other. The code also specifies the way an application requests services from the operating system (OS) or other applications.

g. Beta testing

Beta Testing is one of the Acceptance Testing types, which adds value to the product as the end-user (intended real user) validates the product for functionality, usability, reliability, and compatibility.
Inputs provided by the end-users help in enhancing the quality of the product further and lead to its success. This also helps in decision making to invest further in future products or the same product for improvisation.
Since Beta Testing happens at the end user's side, it cannot be a controlled activity.

h. Alpha testing

Alpha testing is conducted in the organization and tested by a representative group of end-users at the developer's side and sometimes by an independent team of testers.
Alpha testing is simulated or real operational testing at an in-house site. It comes after the unit testing, integration testing, etc. Alpha testing used after all the testing are executed.
It can be a white box, or Black-box testing depends on the requirements - particular lab environment and simulation of the actual environment required for this testing.

I. Integration testing.

Integration testing -- also known as integration and testing (I&T) -- is a type of software testing in which the different units, modules or components of a software

application are tested as a combined entity. However, these modules may be coded by different programmers.

The aim of integration testing is to test the interfaces between the modules and expose any defects that may arise when these components are integrated and need to interact with each other