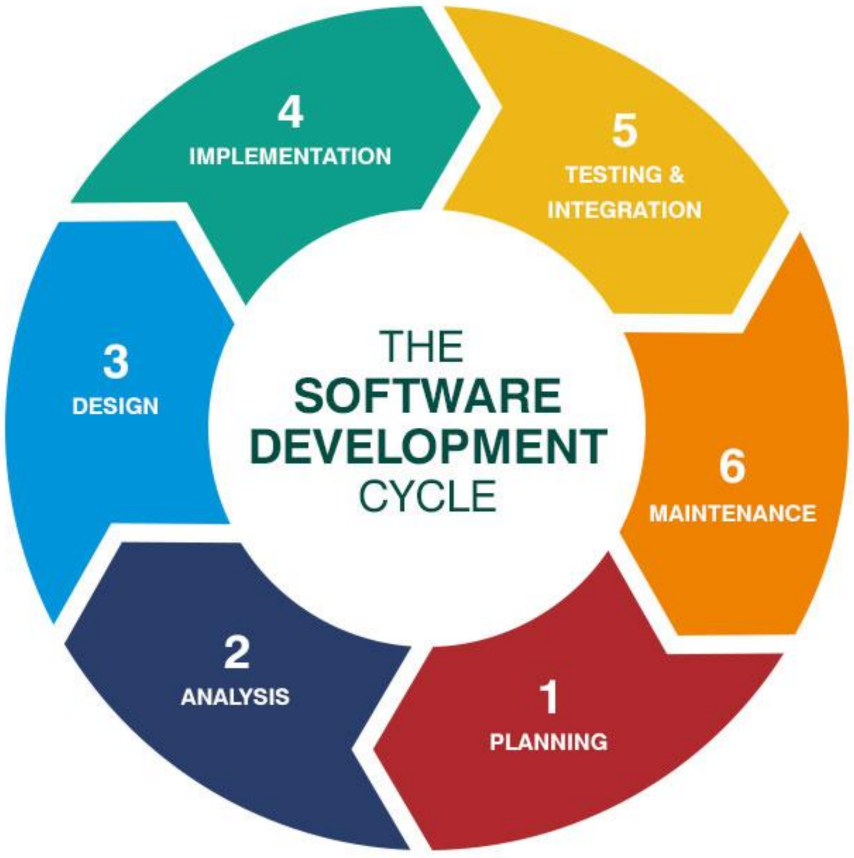
STQA CAT1 QB – Answers

1. a) What is SDLC? Describe in detail.

Ans 1. a)

SDLC stands for "Software Development Life Cycle" it describes the sequence of phases pr steps to develop any software.
In simple words, we can say "entire lifetime of software from beginning to ending".



Stage 1: Planning and Requirement Analysis
Requirement analysis is the most important and fundamental stage in SDLC. It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry. This information is then used to plan the basic project approach and to conduct product feasibility study in the economical, operational and technical areas.

Planning for the quality assurance requirements and identification of the risks associated with the project is also done in the planning stage. The outcome of the technical feasibility study is to define the various technical approaches that can be followed to implement the project successfully with minimum risks.

Stage 2: Defining Requirements
Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysts. This is done through an SRS (Software Requirement Specification) document which consists of all the product requirements to be designed and developed during the project life cycle.

Stage 3: Designing the Product Architecture
SRS is the reference for product architects to come out with the best architecture for the product to be developed. Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification.

This DDS is reviewed by all the important stakeholders and based on various parameters as risk assessment, product robustness, design modularity, budget and time constraints, the best design approach is selected for the product.

A design approach clearly defines all the architectural modules of the product along with its communication and data flow representation with the external and third party modules (if any). The internal design of all the modules of the proposed architecture should be clearly defined with the minutest of the details in DDS.

Stage 4: Building or Developing the Product
In this stage of SDLC the actual development starts and the product is built. The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle.

Developers must follow the coding guidelines defined by their organization and programming tools like compilers, interpreters, debuggers, etc. are used to generate the code. Different high level programming languages such as C, C++, Pascal, Java and PHP are used for coding. The programming language is chosen with respect to the type of software being developed.

Stage 5: Testing the Product
This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC. However, this stage refers to the testing only stage of the product where product defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS.

Stage 6: Deployment in the Market and Maintenance
Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometimes product deployment happens in stages as per the business strategy of that organization. The product may first be released in a limited segment and tested in the real business environment (UAT- User acceptance testing).

Then based on the feedback, the product may be released as it is or with suggested enhancements in the targeting market segment. After the product is released in the market, its maintenance is done for the existing customer base.

1. b) What are the different practices for designing good test case? Explain in detail.

Ans 1. b)

Best practices to write a test case :

Test cases which are easy to execute are considered as good test cases. They make the testing process more efficient by saving time and effort.

We can achieve this by following these 10 best practices :

1) Keeping it Simple and Easy to Understand

A good test case is the one which is easy to understand and execute for the testers. To be a good test case, it should be simple and organized category-wise. Different grouping techniques could be splitting the test cases based on user story or modules like browser specific behaviours etc. This makes it easy to review and maintain. Information given in the test cases should be clear to testers, developers and other stakeholders involved in the project.

2) Including End User Perspective

The end user perspective is a key element when it comes to maintaining the quality of software. Therefore, before drafting a test case, it's important to think like an end user. Understanding the requirement and the functionality aspects covered by the end user's perspective will help in identifying test scenarios that arise in real life business conditions.

3) Using Correct Naming Conventions

Naming the test cases in a way that makes it easy for stakeholders to identify as it will help create good and readable test cases. These also help in traceability as per requirements.

4) Providing Test Case Description

A proper test case description will allow users to understand what is being tested and how. Relevant details like the test environment, test data and tools to be used should also be provided.

5) Including Assumptions

All the assumptions and preconditions that are applicable to the test case should be specified clearly in the test case document.

6) Providing Steps Involved

Steps involved to test a test case should be clearly specified so that if any other person performs the test, they would be clear about what all steps to follow.

7) Giving Details of Test Data

The details of the test data for execution of the test case especially in cases where the same data can be reused helps in saving time for the creation of the test data for each cycle to be run. Aim for maximum coverage by choosing a few select values from each equivalence class.

8) Making it Reusable
Ensuring that there is no dependency or conflict among test cases helps in making it modular. In case there are test cases that are inter-dependent, it should be clearly mentioned in the test document.

9) Assign Testing Priority
Priority of different features in an application is different and therefore assigning a testing priority ensures that during execution, high priority test cases are executed first.

10) Provide The Expected Result and Post Conditions

Expected Results and post conditions help in deciding whether a test case is passed or failed as per the user's acceptance so these should be clearly mentioned in test cases.

The entire testing process becomes more effective when proper time and efforts are invested in creating the test cases thus ensuring the success of the testing plan for the software project!

2. a) Difference between Verification and Validation.

Ans 2. a)

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

2. b) What is test case? Design the test case to for the scenario: “Check Login Functionality”

Ans 2. b)

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.

Test scenarios are rather vague and cover a wide range of possibilities. Testing is all about being very specific.

For a Test Scenario: Check Login Functionality there many possible test cases are:

Test Case 1: Check results on entering valid User Id & Password
Test Case 2: Check results on entering Invalid User ID & Password
Test Case 3: Check response when a User ID is Empty & Login Button is pressed, and many more

The format of Standard Test Cases
Below is a format of a standard login Test cases example.

Test Case ID	Test Case Description	Test Steps	Test Data	Expected Results

TU01	Check Customer Login with valid Data	1. Go to site http://demo.guru99.com
		2. Enter UserId
		3. Enter Password
		4. Click Submit
			Userid = guru99
			Password = pass99
				User should Login into an application

TU02	Check Customer Login with invalid Data	1. Go to site http://demo.guru99.com
		2. Enter UserId
		3. Enter Password
		4. Click Submit
			Userid = guru99
			Password = glass99
				User should not Login into an application

3. a) Differentiate between black box testing and white box testing.

Ans 3. a)

S. No.	Black Box Testing	White Box Testing
1.	It is a way of software testing in which the internal structure or the program or the code is hidden and nothing is known about it.	It is a way of testing the software in which the tester has knowledge about the internal structure or the code or the program of the software.
2.	Implementation of code is not needed for black box testing.	Code implementation is necessary for white box testing.
3.	It is mostly done by software testers.	It is mostly done by software developers.
4.	No knowledge of implementation is needed.	Knowledge of implementation is required.
5.	It can be referred to as outer or external software testing.	It is the inner or the internal software testing.
6.	It is a functional test of the software.	It is a structural test of the software.
7.	This testing can be initiated based on the requirement specifications document.	This type of testing of software is started after a detail design document.
8.	No knowledge of programming is required.	It is mandatory to have knowledge of programming.
9.	It is the behaviour testing of the software.	It is the logic testing of the software.
10.	It is applicable to the higher levels of testing of software.	It is generally applicable to the lower levels of software testing.
11.	It is also called closed testing.	It is also called as clear box testing.
12.	It is least time consuming.	It is most time consuming.
13.	Example: Search something on Google by using keywords	Example: By input to check and verify loops

3. b) What is boundary value analysis? Explain in detail with suitable example.

Ans 3. b)

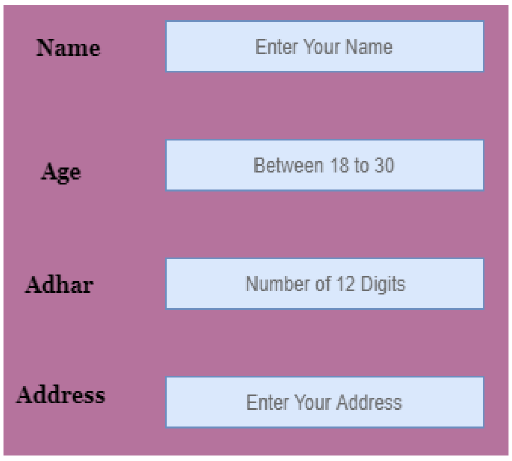
Boundary value analysis is one of the widely used case design technique for black box testing. It is used to test boundary values because the input values near the boundary have higher chances of error.

Whenever we do the testing by boundary value analysis, the tester focuses on, while entering boundary value whether the software is producing correct output or not.

Boundary values are those that contain the upper and lower limit of a variable.

Assume that, age is a variable of any function, and its minimum value is 18 and the maximum value is 30, both 18 and 30 will be considered as boundary values.

The basic assumption of boundary value analysis is, the test cases that are created using boundary values are most likely to cause an error.



Name	Enter Your Name
Age	Between 18 to 30
Adhar	Number of 12 Digits
Address	Enter Your Address

There is 18 and 30 are the boundary values that's why tester pays more attention to these values, but this doesn't mean that the middle values like 19, 20, 21, 27, 29 are ignored. Test cases are developed for each and every value of the range.

Testing of boundary values is done by making valid and invalid partitions. Invalid partitions are tested because testing of output in adverse condition is also essential.

Example:

Imagine, there is a function that accepts a number between 18 to 30, where 18 is the minimum and 30 is the maximum value of valid partition, the other values of this partition are 19, 20, 21, 22, 23, 24, 25, 26, 27, 28 and 29. The invalid partition consists of the numbers which are less than 18 such as 12, 14, 15, 16 and 17, and more than 30 such as 31, 32, 34, 36 and 40. Tester develops test cases for both valid and invalid partitions to capture the behavior of the system on different input conditions.

```
12  14   15   16  17  18  20 22  24  25 26  28  30  31  32  34  36  38  40
--------------------------------|--------------------------------|----------------------------
Invalid Partition               Valid Partition            Invalid Partition
```

Invalid test cases	Valid test cases	Invalid test cases
11, 13, 14, 15, 16, 17	18, 19, 24, 27, 28, 30	31, 32, 36, 37, 38, 39

The software system will be passed in the test if it accepts a valid number and gives the desired output, if it is not, then it is unsuccessful. In another scenario, the software system should not accept invalid numbers, and if the entered number is invalid, then it should display error massage.

If the software which is under test, follows all the testing guidelines and specifications then it is sent to the releasing team otherwise to the development team to fix the defects.

4. a) Enlist any four tools to perform White box testing? Also explain advantages and disadvantages of white box testing.

Ans 4. a)

White Box Testing Tools

#1) Veracode

Veracode's white box testing tools will help you in identifying and resolving the software flaws quickly and easily at a reduced cost. It supports several application languages like .NET, C++, JAVA etc. and also enables you to test the security of desktop, web as well as mobile applications. Still, there are several other benefits of Veracode tool.

#2) EclEmma

EclEmma was initially designed for test runs and analysis within the Eclipse workbench. It is considered to be a free Java code coverage tool and has several features as well.

#3)RCUNIT

A framework which is used for testing C programs is known as RCUNIT. RCUNIT can be used accordingly based on the terms of the MIT License. It is free to use and install.

#4) cfix

cfix is one of the unit testing frameworks for C/C++ which solely aims at making test suites development as simple and easy as possible. Meanwhile, cfix is typically specialized for NT Kernel mode and Win32.

#5) Google test

Google test is Google's C++ test framework. Test Discovery, Death tests, Value-parameterized tests, fatal & non-fatal failures, XML test report generation etc. are few features of Google Test but there are several other features too. Linux, Windows, Symbian, Mac OS X are few platforms where Google Test has been used.

4. b) What is Equivalence class partitioning? Explain in detail with suitable example.

Ans 4. b)

Equivalence class partitioning is a black-box testing technique or specification-based testing technique in which we group the input data into logical partitions called equivalence classes.

All the data items lying in an equivalence class are assumed to be processed in the same way by the software application to be tested when passed as input.

So, instead of testing all the combinations of input test data, we can pick and pass any of the test data from a particular equivalence class to the application and assume that the application will behave in the same way for the other test data of that class. Let's understand this with the help of an example.

Example:

Consider an example of an application that accepts a numeric number as input with a value between 10 to 100 and finds its square. Now, using equivalence class testing, we can create the following equivalence classes-

Equivalence Class	Explanation
Numbers 10 to 100	This class will include test data for a positive scenario.
Numbers 0 to 9	This class will include test data that is restricted by the application. Since it is designed to work with numbers 10 to 100 only.
Greater than 100	This class will again include test data that is restricted by the application but this time to test the upper limit.
Negative numbers	Since negative numbers can be treated in a different way so, we will create a different class for negative numbers in order to check the robustness of the application.
Alphabets	This class will be used to test the robustness of the application with non-numeric characters.
Special characters	Just like the equivalence class for alphabets, we can have a separate equivalence class for special characters.

5. a) What is unit testing? Explain in detail with suitable example.

Ans 5. a)

- **Unit testing is testing the smallest testable unit of an application.**
- **It is done during the coding phase by the developers.**
- **To perform unit testing, a developer writes a piece of code (unit tests) to verify the code to be tested (unit) is correct.**

A Real-world Example

You have written a function to add two numbers:

```
int Add(int a, int b) { return a+b; }
```

The above function takes two numbers as input and returns their sum.

A unit test code would look something like this:

```
void TestAdd1() { Assert.IsEqual(Add(5, 10), 15) }
```

The above unit test “asserts” that 5 + 10 is equal to 15. If the Add function returns anything else Assert.IsEqual result in error and the test case will fail.

You will probably add a few more unit test cases like these:

```
void TestAdd2() { Assert.IsEqual(Add(500, 1000), 1500) }
void TestAdd3() { Assert.IsEqual(Add(0, 1000), 1000) }
void TestAdd4() { Assert.IsEqual(Add(-100, 100), 0) }
void TestAdd5() { Assert.IsEqual(Add(-100, -1100), -1200) }
```

After you write your test cases, you will run them to verify that everything is working correctly.

Later another developer, in addition to adding some of his code, accidentally modifies the Add function as:

```
int Add(int a, int b) { return a*b; }
```

As you can see, instead of adding the two numbers, the code now multiplies them.

When the developer runs the unit tests that you have designed for this function, they will fail. The new developer can trace the failed test cases back to the function and fix the code.

5. b) What is integration testing? Explain in detail with suitable example.

Ans 5. b) Integration testing also known as integration and testing 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.

Integration testing example

Consider a video-streaming mobile application. Its core features include the following:

- Sign up/log in.
- View different monthly/yearly subscription plans.
- Choose personalized plans.
- Watch streaming video.

Once users download the applications, they see a sign-up form where they can enter their account information. After successful authorization, they are redirected to a page listing different subscription plans. They can choose their own plan and then complete the payment.

Any errors in this logical flow could cause problems for the user and lead to losses for the app company. Integration testing can help find and fix such errors.
So, after each module is ready, testers conduct unit testing. And once all modules are available, testers test them together to check their interfaces and data flows. If no errors are detected, the end user should be able to successfully complete their transaction.

6. a) Explain in detail about advantages and disadvantages of unit testing.

Ans 6. a)

Advantages of Unit Testing:

1. **Unit Testing allows developers to learn what functionality is provided by a unit and how to use it to gain a basic understanding of the unit API.**
2. **Unit testing allows the programmer to refine code and make sure the module works properly.**
3. **Unit testing enables testing parts of the project without waiting for others to be completed.**
4. **Early Detection of Issues: Unit testing allows developers to detect and fix issues early in the development process, before they become larger and more difficult to fix.**
5. **Improved Code Quality: Unit testing helps to ensure that each unit of code works as intended and meets the requirements, improving the overall quality of the software.**
6. **Increased Confidence: Unit testing provides developers with confidence in their code, as they can validate that each unit of the software is functioning as expected.**
7. **Faster Development: Unit testing enables developers to work faster and more efficiently, as they can validate changes to the code without having to wait for the full system to be tested.**
8. **Better Documentation: Unit testing provides clear and concise documentation of the code and its behavior, making it easier for other developers to understand and maintain the software.**
9. **Facilitation of Refactoring: Unit testing enables developers to safely make changes to the code, as they can validate that their changes do not break existing functionality.**
10. **Reduced Time and Cost: Unit testing can reduce the time and cost required for later testing, as it helps to identify and fix issues early in the development process.**

Disadvantages of Unit Testing:

1. **The process is time-consuming for writing the unit test cases.**
2. **Unit Testing will not cover all the errors in the module because there is a chance of having errors in the modules while doing integration testing.**
3. **Unit Testing is not efficient for checking the errors in the UI(User Interface) part of the module.**
4. **It requires more time for maintenance when the source code is changed frequently.**
5. **It cannot cover the non-functional testing parameters such as scalability, the performance of the system, etc.**
6. **Time and Effort: Unit testing requires a significant investment of time and effort to create and maintain the test cases, especially for complex systems.**
7. **Dependence on Developers: The success of unit testing depends on the developers, who must write clear, concise, and comprehensive test cases to validate the code.**
8. **Difficulty in Testing Complex Units: Unit testing can be challenging when dealing with complex units, as it can be difficult to isolate and test individual units in isolation from the rest of the system.**
9. **Difficulty in Testing Interactions: Unit testing may not be sufficient for testing interactions between units, as it only focuses on individual units.**
10. **Difficulty in Testing User Interfaces: Unit testing may not be suitable for testing user interfaces, as it typically focuses on the functionality of individual units.**
11. **Over-reliance on Automation: Over-reliance on automated unit tests can lead to a false sense of security, as automated tests may not uncover all possible issues or bugs.**
12. **Maintenance Overhead: Unit testing requires ongoing maintenance and updates, as the code and test cases must be kept up-to-date with changes to the software.**

6. b) Explain in detail about Acceptance Testing.

Ans 6. b)

Acceptance testing is a phase after system testing that is normally done by the customers or representative of the customer. Acceptance test is performed by the client, not by the developer. Acceptance test cases are normally small in number and are not written with the intention of finding defects. However, the purpose of this test is to enable customers and users to determine if the system built really meets their needs and expectations. Acceptance testing is done by the customer or by the representative of the customer to check whether the product is ready for use in the real-life environment.

Acceptance tests are written to execute near real-life scenarios.

Acceptance Criteria

1. Product acceptance
2. Procedure acceptance
3. Service level agreements

Selection test cases for Acceptance Testing

- End-to-end functionality verification
- Domain tests
- User scenario tests
- Basic sanity tests
- New functionality
- A few non-functionality
- Tests pertaining to legal obligations and service level agreements
- Acceptance test data

METHOD

Usually, Black Box Testing method is used in Acceptance Testing. Testing does not normally follow a strict procedure and is not scripted but is rather ad-hoc.

TASKS

1. Acceptance Test Plan
   - I. Prepare
   - II. Review
   - III. Rework
   - IV. Baseline
2. Acceptance Test Cases/Checklist
   - I. Prepare
   - II. Review
   - III. Rework
   - IV. Baseline
3. Acceptance Test
   - Perform

When is it performed?
Acceptance Testing is performed after System Testing and before making the system available for actual use.

Who performs it?

- **Internal Acceptance Testing (Also known as Alpha Testing) is performed by members of the organization that developed the software but who are not directly involved in the project (Development or Testing). Usually, it is the members of Product Management, Sales and/or Customer Support.**
- **External Acceptance Testing is performed by people who are not employees of the organization that developed the software.**
    - **Customer Acceptance Testing is performed by the customers of the organization that developed the software. They are the ones who asked the organization to develop the software. [This is in the case of the software not being owned by the organization that developed it.]**
    - **User Acceptance Testing (Also known as Beta Testing) is performed by the end users of the software. They can be the customers themselves or the customers‘ customers.**

Types of Acceptance testing

- **Benchmarking: a predetermined set of test cases corresponding to typical usage conditions is executed against the system**
- **Pilot Testing: users employ the software as a small-scale experiment or in a controlled environment**
- **Alpha-Testing: pre-release closed / in-house user testing**
- **Beta-Testing: pre-release public user testing**
- **Parallel Testing: old and new software are used together and the old software is gradually phased out.**

7. a) What is Test Data Generation? Why test data should be created before test execution?

Ans 7. a)

Everybody knows that testing is a process that produces and consumes large amounts of data. Data used in testing describes the initial conditions for a test and represents the medium through which the tester influences the software. It is a crucial part of most Functional Tests.

Depending on your testing environment you may need to CREATE Test Data (Most of the times) or at least identify a suitable test data for your test cases (is the test data is already created).
Typically test data is created in-sync with the test case it is intended to be used for.
Test Data can be Generated –

- **Manually**
- **Mass copy of data from production to testing environment**
- **Mass copy of test data from legacy client systems**
- **Automated Test Data Generation Tools**

Typically, sample data should be generated before you begin test execution because it is difficult to handle test data management otherwise. Since in many testing environments creating test data takes multiple pre-steps or very time-consuming test environment configurations.

Also, if test data generation is done *while* you are in test execution phase you may exceed your testing deadline.

7. b) Discuss about the various challenges in test automation.

Ans 7. b)

There are 7 challenges in Test Automation -

1. Team Collab Issues

It is necessary to keep your team in a loop and to ensure that they all are in the same alignment. With the growing age of technology, firms are now moving towards automation and that has become a challenge for QAs. The reason being they do lack a skill set and are not much well aware of the backend codes. Thus, this becomes very challenging for them to work with the script.

Therefore, this issue can be solved by keeping your whole team in a loop and making sure that they all are on the same path where they can exchange information and communicate as per their needs. Besides, some other practices can be also be taken into considerations:

- **By providing proper training so that they can be well aware of the outlook.**
- **By creating a common communication channel.**
- **Take feedbacks about the tasks that they don't feel good about.**

2. Network Issues

During the automation testing, it is being observed that network stability & disconnection has been one of the most challenging issues for QAs. The dedicated team needs to have a good and stable connection, and if the connection is poor, it becomes hard to access databases, third-party services, etc.

Some other issue that occurs with the network is:

- **Disruption in the network also blocks QA to access virtual environments that are used for testing.**
- **Any delay in testing, specifically due to network issues can cause delays in the whole development cycle.**
- **In the development cycle, the whole system fails if testing goes wrong. This happens due to its architecture, and there are many testing environments that run on a single point.**

3. Cost Factor

You might not be surprised that test automation has become a very crucial part of software testing. Besides automation comes at a costly price especially during its initial phase. So, if you're running on a tight budget, you might need to reconsider it again because the implementation is going to cost you on a higher side which also includes license cost for the software. Likewise, going for an open-source solution will lead you to train and maintenance costs, which makes it even from all aspects so make sure you're pitching correctly to the management to plan for it.

4. Right Expectations
It is a must to understand that expectations should always be right either it's for selecting testing tools or the methods you're going to implement or maybe the outcome you're expecting. No matter how much automation we're going to make but human interaction and involvement are necessary and will play a very crucial role in fixing bugs and running tests.
Therefore, for non-automation testing, humans will be required, and not to forget the most important thing is *'A testing phase gets success not because of the number of executions, but its success defines by the number of successful compilations done with a meaningful outcome.'*
So, make sure that you're getting the right expectation with stakeholders and management so that they can plan accordingly.

5. Testing Accuracy
If you try to run processes on outdated data, then it will show you the false outputs which are the most common mistakes in automation. That's why it is very crucial for a team to be aligned in a singular channel which can help them in having clear communication to avoid any such chaos. In Automation, DevOps needs a quick response from testers in order to get more accuracy over the data. Thus, data relevancy and accuracy are the only factors to achieve in testing accuracy.
On the other hand, it's also required for QAs to work on a robust analytical solution to enhance productivity.

6. Selection of Tool
The selection of the right tool is the most crucial part of the testing phase. Lack of expertise can hamper the whole development cycle. Generally, typical testing consists of regression, functional, unit, integration, etc. Besides the fact, the challenging part is to decide whether your project needs Automation or Manual Testing. Make sure to dig into these considerations before selecting any tool for your project:

- **Gather information about your project and its requirement.**
- **Technical support & assistance.**
- **Check for cross-browsing testing compatibility.**
- **Ease in maintenance**
- **Cost**

7. Fixing Code smell
It is very important to address and fix up a bad code smell from any frame. Why is it so crucial to fix code smell? A Code smell is a part of a code that is not "technically wrong" but its implementation is chaotic and becomes a pain during the execution. So, it's a responsibility for both programmers and testers to take care of the quality of code so that it doesn't affect the development cycle.
However, it will also lead to maintaining a pace in project development and makes automation more robust.

8. a) What is Automation Testing? Explain in detail with suitable example.

Ans 8. a)

Automation Testing is a software testing technique that performs using special automated testing software tools to execute a test case suite. On the contrary, Manual Testing is performed by a human sitting in front of a computer carefully executing the test steps.

Examples

1. **While these tests can be performed by a human, they are quite complex and are therefore prone to errors.**
   For example, someone testing a site in a foreign language is bound to make mistakes, especially if the site is sizable. In instances like this, it's easy to see why automation testing is the right option.
   a. **That said, there are some instances where manual testing is better, including:**
   b. **New test cases that have not yet been executed manually**
   c. **Test cases where the criteria are always changing**
   d. **Test cases that are not routine**

2. **In these instances, you can see why it would be beneficial to have a pair of human eyes on the testing.**

   For example, the first time a test code is written, it should be run manually to ensure that it delivers the expected result. Once this is verified, it can then be used as an automated solution.

   In the cases where automation testing is appropriate, you'll see some specific benefits, (perhaps even more so if you are already using AI in test automation) including:
   a. **Speed**
   b. **Wider test coverage**
   c. **Consistency**
   d. **Cost savings**
   e. **Frequent and thorough testing**
   f. **Faster time to market**

8. b) Explain the test plan with proper scenario.

Ans 8. b)

A Test Plan is a detailed document that describes the test strategy, objectives, schedule, estimation, deliverables, and resources required to perform testing for a software product. Test Plan helps us determine the effort needed to validate the quality of the application under test. The test plan serves as a blueprint to conduct software testing activities as a defined process, which is minutely monitored and controlled by the test manager.

As per ISTQB definition: "Test Plan is A document describing the scope, approach, resources, and schedule of intended test activities."

Let's start with following Test Plan example/scenario: In a meeting, you want to discuss the Test Plan with the team members, but they are not interested – .

How to write a Test Plan

You already know that making a Test Plan is the most important task of Test Management Process. Follow the seven steps below to create a test plan as per IEEE 829

- **I. Analyze the product**
- **II. Design the Test Strategy**
- **III. Define the Test Objectives**
- **IV. Define Test Criteria**
- **V. Resource Planning**
- **VI. Plan Test Environment**
- **VII. Schedule & Estimation**
- **VIII. Determine Test Deliverables**

Step 1) Analyze the product

How can you test a product without any information about it? The answer is Impossible. You must learn a product thoroughly before testing it.
The product under test is Guru99 banking website. You should research clients and the end users to know their needs and expectations from the application

- **Who will use the website?**
- **What is it used for?**
- **How will it work?**
- **What are software/ hardware the product uses?**

Step 2) Develop Test Strategy

Test Strategy is a critical step in making a Test Plan in Software Testing. A Test Strategy document, is a high-level document, which is usually developed by Test Manager. This document defines:

- **The project's testing objectives and the means to achieve them**
- **Determines testing effort and costs**

Step 2.1) Define Scope of Testing
Step 2.2) Identify Testing Type
Step 2.3) Document Risk & Issues
Step 2.4) Create Test Logistics

Step 3) Define Test Objective

Test Objective is the overall goal and achievement of the test execution. The objective of the testing is finding as many software defects as possible; ensure that the software under test is bug free before release.
To define the test objectives, you should do 2 following steps

- **List all the software features (functionality, performance, GUI…) which may need to test.**
- **Define the target or the goal of the test based on above features**

Step 4) Define Test Criteria

Test Criteria is a standard or rule on which a test procedure or test judgment can be based.

There're 2 types of test criteria as following

I. **Suspension Criteria**
II. **Exit Criteria**

Test Plan Example: Your Team has already done the test executions. They report the test result to you, and they want you to confirm the Exit Criteria.

Step 5) Resource Planning

Resource plan is a detailed summary of all types of resources required to complete project task. Resource could be human, equipment and materials needed to complete a project

The resource planning is important factor of the test planning because helps in determining the number of resources to be used for the project. Therefore, the Test Manager can make the correct schedule & estimation for the project.

Step 6) Plan Test Environment

A testing environment is a setup of software and hardware on which the testing team is going to execute test cases. The test environment consists of real business and user environment, as well as physical environments, such as server, front end running environment.

Step 7) Schedule & Estimation

In the article Test estimation, you already used some techniques to estimate the effort to complete the project. Now you should include that estimation as well as the schedule to the Test Planning

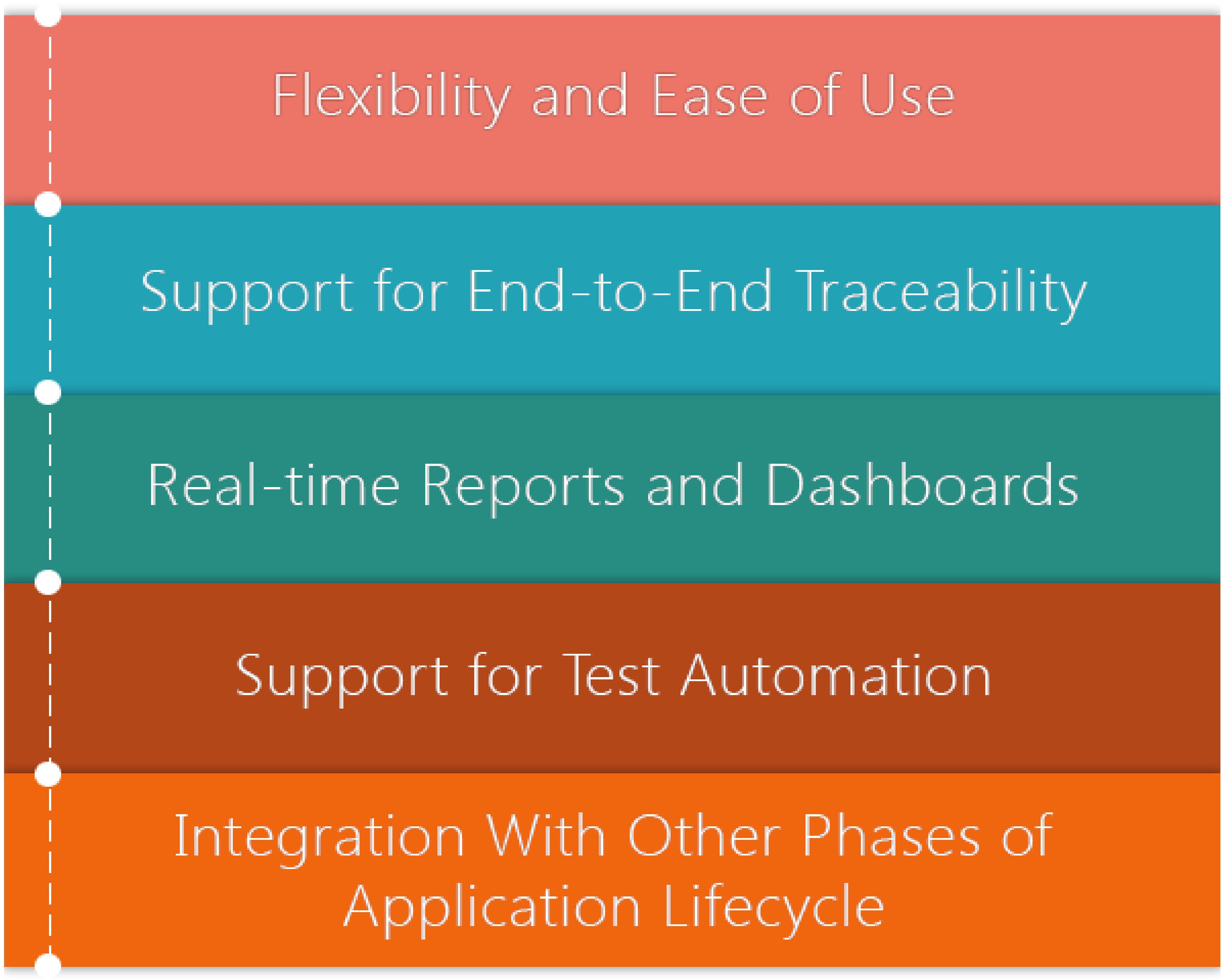
Step 8) Test Deliverables

Test Deliverables is a list of all the documents, tools and other components that has to be developed and maintained in support of the testing effort.
There are different test deliverables at every phase of the software development lifecycle.

9. a) Enlist criteria for selecting a testing tool.

Ans 9. a)

5 Major Criteria for Selecting a Testing Tool


Flexibility and Ease of Use
Support for End-to-End Traceability
Real-time Reports and Dashboards
Support for Test Automation
Integration With Other Phases of Application Lifecycle

1. **Flexibility and Ease of Use**

In the absence of a proper tool, MS Excel is the most popular offline medium for managing test artifacts. One of the major reasons for this is the ease of use and flexibility of the tool. So, while moving from Excel to a standard test tool, it has to be made sure that the tool is very easy to use and its training and user adaptation time is very less. Every organization, and in some cases every project follows their own model of testing. The testing tool should be configurable enough to support these model variances.

2. **Support for End-to-End Traceability**

It is very important for the testers to able to trace back all their work in a centralized test management system. A bi-directional traceability between test artifacts and the associated requirements and defects increases the efficiency of measuring quality of a project. It also allows organizations to track the coverage of both Requirements and Test Cases, failing which may lead to missing information, loss of productivity and fall in quality.

3. Real-time Reports and Dashboards

   Most software projects fail due to the lack of proper visualization of analytical data related to a project's progress. In the absence of a centralized tool, the entire process of reporting is dependent on manual interactions, making it error-prone. So, the tool to be procured should have the facility of providing real-time reports and dashboards, keeping stakeholders updated with the latest status of progress and assess quality at every step.

4. Support for Test Automation

   Today, due to fierce market competition, Test Automation is no more a choice but has become a mandate for organizations. A testing tool must have the support for managing Test Automation scripts from a single repository. However, that is not enough for a Test automation project. Test automation needs to be an integral part of the entire execution process. Features like the central execution of test automation scripts, automatic capturing of test results and making them visible from a central platform are necessary. Viewing Test Automation results needs to be a part of the end-to-end traceability chain.

5. Integration With Other Phases of Application Lifecycle

   Today, in the world of Agility, testing is no longer an isolated phase or a security gate to final delivery, but an integral part of the entire lifecycle. To achieve this and ensure quality right from the beginning, testing should get involved at every stage of the lifecycle. Therefore, a testing tool should have the capability to integrate with tools from other phases of the lifecycle, so that a centralized status update on the project's progress and quality can be achieved. Seamless integration between testing and other lifecycle tools paves the way for an organization to achieve Continuous Integration (CI) and Continuous Delivery (CD), the milestones to implement DevOps.

9. b) Explain in detail about web testing with suitable example.

Ans 9. b)

Web testing, at its core, is simply checking your web application or your website for problems before you make that web application or website live. Web testing is designed to check all aspects of the web application's functionality, including looking for bugs with usability, compatibility, security, and general performance. Web testing is a crucial part of assembling any web application or website, as you don't want to invest the many resources in time and money you've spent developing this web application and then have it run into immediate problems upon release. We have seen that happen before, and it isn't pretty.

Types of Web Application Testing

1. **Testing the Functionality and Features**
2. **Testing Web APIs**
3. **Testing the Database**

Example of a web testing a web application:

1. **Checking that the login to your web application is successful**
2. **Checking that the login to your web application is successful across browsers and devices**
3. **Forms with multiple fields are populated with the right data**
4. **The web application is interacting with external databases, and syncing successfully**
5. **Invoices are being sent and received with the correct information and securely**
6. **The functionality of buttons across pages are working as per the requirements**

10. a) What Is syntax testing? Discuss in detail about its formats and test cases.

Ans 10. a)

Syntax Testing, a black box testing technique, involves testing the System inputs and it is usually automated because syntax testing produces a large number of tests. Internal and external inputs have to conform the below formats:

- **Format of the input data from users.**
- **File formats.**
- **Database schemas.**

Syntax Testing - Steps:

i. **Identify the target language or format.**
ii. **Define the syntax of the language.**
iii. **Validate and debug the syntax.**

Syntax Testing - Limitations:

i. **Sometimes it is easy to forget the normal cases.**
ii. **Syntax testing needs driver program to be built that automatically sequences through a set of test cases usually stored as data.**

Syntax Testing – Test Cases:

The preferred test strategy which may be followed up to perform syntax testing is to create one error in the input string, one at time while keeping all other components of the string unchanged and correct. Thereafter, the said process needs to repeat until the complete set of test for the single errors get defined. Similarly, the complete set of test for the double error, triple error and so on may be defined and designed. During the whole process, the focus should be at one level, along with the correctness both at its upper and lower level.

10. b) Define object oriented testing? Explain its related issues.

Ans 10. b)

Software typically undergoes many levels of testing, from unit testing to system or acceptance testing. Typically, in-unit testing, small “units”, or modules of the software, are tested separately with focus on testing the code of that module. In higher, order testing (e.g., acceptance testing), the entire system (or a subsystem) is tested with the focus on testing the functionality or external behavior of the system.

As information systems are becoming more complex, the object-oriented paradigm is gaining popularity because of its benefits in analysis, design, and coding. Conventional testing methods cannot be applied for testing classes because of problems involved in testing classes, abstract classes, inheritance, dynamic binding, message, passing, polymorphism, concurrency, etc.

Testing classes is a fundamentally different problem than testing functions. A function (or a procedure) has a clearly defined input-output behavior, while a class does not have an input-output behavior specification. We can test a method of a class using approaches for testing functions, but we cannot test the class using these approaches.

According to Davis the dependencies occurring in conventional systems are:

- **Data dependencies between variables**
- **Calling dependencies between modules**
- **Functional dependencies between a module and the variable it computes**
- **Definitional dependencies between a variable and its types.**

But in Object-Oriented systems there are following additional dependencies:

- **Class to class dependencies**
- **Class to method dependencies**
- **Class to message dependencies**
- **Class to variable dependencies**
- **Method to variable dependencies**
- **Method to message dependencies**
- **Method to method dependencies**