QA General Answer Sheet of Bashiul Alam Sabab

Question:

How would you define a ‘test plan’? What are the aspects that all test plan should contain?

Answer:

Test Plan is a document which outlines the details of the whole test strategy and objectives. This document is a detailed piece of documentation which also covers what are included and excluded in our testing scope, what are the exit criteria of test completion and deliverables during the whole testing lifecycle.

A Test Plan contains some additional sections apart from the ones stated above. All of these are listed below with their corresponding description on what to include in the documentation.

Section	Subsection	Description
Introduction		A briefing on the purpose of the test plan document
Terms/Acronyms		Details of the technical terminologies used throughout the document and abbreviations details
Test Strategy	Scope	Which types of tests and test methodologies that are included and excluded within the testing scope
	Risk and Issues	What are the risks and challenges (both technical and non-technical) to consider within the testing lifecycle and how to overcome or mitigate them
	Test Logistics	The details of the person(s) or team who will be responsible for testing and defining all the prerequisites of

		testing
Test Objective		Defining the overall objective and what are the target achievements from the testing scope
Test Criteria	Suspension Criteria	Defining the condition based on which test testing process will be suspended until further improvement on code base
	Exit Criteria	Specifying the criteria and the metrics which denote a successful completion of the test phase
	Bug Triage	Defining the process of bug life cycle, which bugs to fix and when to fix them
Resource Planning	System and Physical Resource	Identifying resources like: ● Test Servers and network requirements ● Project Management, Test Management, Bug Tracking and other associated tools ● Automation frameworks and associated tools ● Non-functional testing tools and resources
	Roles and Responsibilities	Defining the roles and responsibilities of each personnel involved in the testing lifecycle
Test Estimation		Estimation of timeline to complete each testing phase mentioned in the "*scope*" section
Test Deliverables	Deliverables before testing starts	- Test plan document - Test cases documents - Test design specifications
	Deliverables during testing phase	- Automation test suite - Test Data - Test Trace-ability Matrix - Error logs and execution logs

	Deliverables after testing phase completion	- Test results/reports - Defect report - Installation/ Test procedures guidelines - Release notes

Question:

How would you write a good bug report that tells the developer what the bug is? Explain with an example.

Answer:

A good bug report should contain all the necessary information regarding what went wrong, where it went wrong.

A bug report should contain the following points:

1. Bug ID: A unique ID to track the bug in the whole bug life cycle. If we are using a bug tracking tool, this ID is generated automatically.
2. Summary: Summary should be a single sentence summarizing the occurred bug in a simple manner.
3. Description: Description section should elaborate the summary line.
4. Steps to Reproduce: The steps taken to produce the bug.
5. Expected Result: What is/are the expected result(s) executing the test steps.
6. Actual Result: What is the outcome or system behavior after executing the test steps.
7. Severity: The impact level the bug has on the application.
8. Priority: The order in which the bug should be considered for fixing and deployment.
9. Component: Component can be a micro-service name or some terminology like if it is a front end or back end bug. If it is a front end bug we can mention if it is occurring on web platform or on mobile app.
10. Test Logs/Screenshot: For back-end bugs we can include application log or browser network log for ease of troubleshooting. For front-end bugs we can add a screenshot or the screen recording of the steps to reproduce.
11. Version of the Application/Affecting Version: We can add the version of the release or web or mobile app version on which we have tested and discovered the bug.
12. Assignee: Assignee should be the corresponding developer of the feature or it can be the dev lead or the product owner.
13. Bug Reporter: Reporter name should be added so that the assigned developer can know whom to contact in case of any need. If we use a bug tracking tool this field is auto generated.

14. Labels: We can add convenient labels so that the bug is easy to track and it becomes easy to create a bug report later on using the label.
15. Test Case Link (Optional): Optionally we can add our test case link with the bug report to make it easy to track the status of the executed test case.

A Sample Bug Report:

Lets say we are testing a mobile wallet application which has a feature of send money to others. After send money operation is complete our mobile app receives a push notification from back-end that whether the transaction was successful or not and based on that app shows a transaction closure pop-up and generates a transaction history entry for this operation. Now lets assume the bug is that our mobile app receives the push notification but failed to show the pop-up and generate the transaction history. Bug report for this situation will be like below:

Section	**Bug Content**
Bug ID	WALLET-101
Summary	[Android App Bug] Application fails to show send money pop-up and does not generate transaction history
Description	After send money is complete, android app is getting push notification from back-end but not showing the transaction closure pop-up and not generating the transaction history for this operation.
Steps to Reproduce	1. Login to app 2. Tap on Send Money 3. Enter destination account number 4. Enter amount 5. Tap on Send button 6. Enter your PIN
Expected Result	1. Send money should be successful 2. App should show a transaction successful pop-up 3. Transaction entry should be available in transaction history
Actual Result	1. App is not showing the transaction successful pop-up 2. Transaction history entry is not generated

Severity	High
Priority	High
Component	Android App
Test Log	● Back-end log that push notification sending was successful ● For debug apk builds we can add the android log if that is available ● Or we can add a screen recording of the whole operation
Version	Android App v1.1.1
Assignee	App Developer
Reporter	Bashiul Alam Sabab
Label	Android_App_Bug, v1.1.1_Bug
Test Case	SendMoney001

Question:

Name at least 5 crucial things while testing a SaaS B2B Platform.

Answer:

1. First of all like all other software delivery models it is essential to test the overall functionality of a SaaS B2B Platform. We need to validate that all the functionalities match our requirements and there is no bug in the basic functional flow of each and every feature.
2. In a subscription based model we need to put extra effort in testing “authentication” and “authorization”. This is important because we need to make sure our platform is granting access to the correct user and customer to the portion where they are entitled to.
3. A typical SaaS platform will handle huge amount of data in a day to day basis. In this regard checking data integrity is crucial. Checking data integrity for each customer base is necessary.
4. As SaaS platform handles huge amount of data, data migration is another aspect besides data integrity where the application should perform smoothly.
5. As SaaS platform provides service over the internet, it is essential to check performance of the platform over different network situations and bandwidth.
6. Scalability is also another important part to validate. The platform should be able to accommodate user base from different geo locations with different size of data.
7. Ensuring overall network and system security is also essential.
8. As part of non-functional testing, performance and load testing is another aspect where the platform should perform within defined benchmark values.

Question:

How would you take care of UI validations during testing?

Answer:

For testers the exit criteria or validation criteria should always be defined based on some source of truth like requirement documents. In terms of UI design and implementation the source of truth should be the design documentation or design guide. In ideal scenario the design document should be ready at least in a draft state before UI/UX development starts. That way the testers or the QA team can prepare some UI test cases to validate the UI functionality.

When we are testing UI, first of all we need to validate the overall functional flow of the application UI. We need to check if the menu items, button items or link items are clickable and on triggered action events they are navigating to correct sources. Then we need to check the different fields in a form. We need to validate the data type of the individual fields, the form validations, field value masking, dropdown actions etc.

Next we need to validate the overall usability of the UI. First thing in testing the usability includes UI labels and text matching with the design documents. We also need to check different user prompt messages and error messages if these are matching with the ones defined in the design guide. We need to make sure there are no typos or grammatical mistakes in them. Next part of usability comes in the form of checking the responsiveness of the UIs. We need to check the application UI in different platforms and screen sizes and resolutions. Lastly, we need to check UI performance in different network connectivity as most of the applications nowadays having a client-server architecture maintain a connection with the internet.

Question:

How would you prioritize test cases in a situation where estimated test effort exceeds the delivery timeline?

Answer:

When we are preparing test cases we should always try to assign priority to them. This will help us when our estimated test effort can exceed the delivery timeline. Meeting deadline is a very important aspect of overall SDLC and STLC. So if we face the situation where our detailed testing cannot be covered within the delivery timeframe we need to prioritize or set up an order of executing our cases based on the following strategy:

1. Execute the basic functional flow coverage test cases which satisfies the product/feature requirements: These cases should be assigned the highest priority as they ensure that our product is functioning as it is supposed to.
2. Execute the cases which were added because of production/live bugs in the past or which were added due to any kind of bug leakage in the past: Usually the bug leakage test cases and scenarios are added in the regression test suite. But before signing off a release it would be helpful and a good idea to make sure these cases pass.
3. Execute the negative test cases: These cases should come in the third order of execution.
4. Execute the edge and corner cases: In ideal cases the edge cases like, Database connection is unavailable, application server has crashed etc. can be considered as the lowest priority and they can be executed at last.

Based on the order above we should plan our test execution timeline so that we can meet the delivery timeline. If the timeline is too tight we need to at least execute the cases from points **1** and **2.** The cases from the other points can be covered after we provide the release if time permits or we can adjust our regression test suite to cover some cases from points **3** and **4** to ensure we have executed most of the test cases and achieved a satisfactory amount of test coverage.