QA Engineering
Software Testing Methodology

Nizam Mahmood

Infrastructure Architect

Software Testing

What is Software Testing?

Software Testing is the process or method of finding error/s in a software application or program so that the application functions according to the end user's requirement.

Why is Software Testing Important?

Testing is important because software bugs could be expensive or even dangerous. Software bugs can potentially cause monetary and human loss.

- ✓ In April 2015, Bloomberg terminal in London crashed due to software glitch affected more than 300,000 traders on financial markets. It forced the government to postpone a 3bn pound debt sale.
- ✓ Nissan cars have to recall over 1 million cars from the market due to software failure in the airbag sensory detectors. There has been reported two accident due to this software failure.
- ✓ Starbucks was forced to close about 60 percent of stores in the U.S and Canada due to software failure in its POS system. At one point store served coffee for free as they unable to process the transaction.

Testing Types

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.

Testing Types

Testing Category	**Types of Testing**
Functional Testing	Unit Testing
	Black box Testing
	White box Testing
	Smoke Testing
	UAT (User Acceptance Testing)
	Sanity Testing
	Sanity Testing
	Integration Testing
Non-Functional Testing	Performance Testing
	Endurance Testing
	Load Testing
	Volume Testing
	Scalability Testing
	Security Testing
Maintenance	Regression
	Maintenance

Testing Methods

- **Black-Box Testing**:
  - Testing technique that calls for the tester to have NO knowledge of the interior workings of the application
  - The tester **does not** know the programming code running in the backend
  - The tester interacts with the front-end of the application (AKA the Graphical User Interface – GUI)
  - Tester provides inputs and examines the outputs without knowing how and where the inputs are worked upon
  - **Advantages of Black-Box Testing:**
    - Well suited and efficient for large code segments.
    - Code access is not required.
    - Clearly separates user's perspective from the developer's perspective through visibly defined roles.
    - Large numbers of moderately skilled testers can test the application with no knowledge of implementation, programming language, or operating systems.
  - **Disadvantages of Black-Box Testing:**
    - Limited coverage, since only a selected number of test scenarios is actually performed.
    - Inefficient testing, due to the fact that the tester only has limited knowledge about an application.
    - Blind coverage, since the tester cannot target specific code segments or error-prone areas.
    - The test cases are difficult to design.

Testing methods *continued*

- **White-Box Testing**:
  - Detailed investigation of internal logic and structure of the code
  - Also called **Glass-Box Testing, Open-Box Testing**
  - In order to perform White-Box Testing, a tester must know the programming code that is running the application
  - The tester must look at the code and determine which unit (chunk) of code is behaving inappropriately
  - <u>Advantages of White-Box Testing:</u>
    - As the tester has knowledge of the source code, it becomes very easy to find out which type of data can help in testing the application effectively.
    - It helps in optimizing the code.
    - Extra lines of code can be removed which can bring in hidden defects.
    - Due to the tester's knowledge about the code, maximum coverage is attained during test scenario writing.
  - <u>Disadvantages of White-Box Testing</u>:
    - Due to the fact that a skilled tester is needed to perform white-box testing, the costs are increased.
    - Sometimes it is impossible to look into every nook and corner to find out hidden errors that may create problems, as many paths will go untested.
    - It is difficult to maintain white-box testing, as it requires specialized tools like code analyzers and debugging tools.

Testing methods *continued*

- **Grey-Box Testing:**
  - It’s a technique to test the application by having a **limited understanding** of the internal workings of the application
  - The tester has access to the system design documents and the database
  - Having this knowledge, a tester can prepare better test data and test scenarios while making the test plan
  - <u>Advantages of Grey-Box Testing</u>:
    - Offers combined benefits of black-box and white-box testing wherever possible.
    - Based on the limited information available, a grey-box tester can design excellent test scenarios especially around communication protocols and data type handling.
    - The test is done from the point of view of the user and not the designer.
  - <u>Disadvantages of Grey-Box Testing:</u>
    - Since the access to code is not readily available, the ability to go over the code and test coverage is limited.
    - The tests can be redundant if the software designer has already run a test case.
    - Testing every possible input stream is unrealistic because it would take an unreasonable amount of time; therefore, many program paths will go untested.

Side-By-Side Comparison

Black-Box Testing	Grey-Box Testing	White-Box Testing
The internal workings of an application need not be known.	The tester has limited knowledge of the internal workings of the application.	Tester has full knowledge of the internal workings of the application.
Also known as closed-box testing, data-driven testing, or functional testing.	Also known as translucent testing, as the tester has limited knowledge of the insides of the application.	Also known as clear-box testing, structural testing, or code-based testing.
Performed by end-users and also by testers.	Performed by end-users and also by testers and developers.	Normally done by developers.
Testing is based on external expectations - Internal behavior of the application is unknown.	Testing is done on the basis of high-level database diagrams and data flow diagrams.	Internal workings are fully known and the tester can design test data accordingly.
It is exhaustive and the least time-consuming.	Partly time-consuming and exhaustive.	The most exhaustive and time-consuming type of testing.
Not suited for algorithm testing.	Not suited for algorithm testing.	Suited for algorithm testing.
This can only be done by trial-and-error method.	Data domains and internal boundaries can be tested, if known.	Data domains and internal boundaries can be better tested.

Testing levels continued

- **Functional Testing**:
  - Type of Black-Box Testing that is based on the requirements of the software that is to be tested
  - The application is tested by providing inputs and then analyzing the outputs and verifying that the expected outputs match the actual outputs
  - Conducted on a complete, integrated system to evaluate the system's compliance with the specified requirements
  - **5 steps involved in functional testing**

  1. The determination of the functionality that the intended application is meant to perform.
  2. The creation of test data based on the specifications of the application.
  3. Determination of the output based on the test data and the specifications of the application
  4. The writing of test scenarios and the execution of test cases.
  5. The comparison of actual and expected results based on the executed test cases.

- **Unit Testing** (Functional Testing)
  - Generally performed by developers before the setup is handed over to the testing team to formally execute test cases
  - Each developer Unit Tests the bit of code they've developed (think of a student proof-reading an essay)
  - Goal of Unit Testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionality
  - This way, if the chunks of code, once put together, do not function properly, the problem lies in the integration, not in the individual units of code
  - Limitations of Unit Testing:
  - Testing can't catch every bug in the application
  - Testing at each level is similar to having various safety nets throughout the development process to catch bugs before the application goes to production

Testing levels *continued*

- **Integration Testing**: (Functional Testing)
  - Testing combined parts of the application to determine if they function correctly
  - Can be done in 2 ways – **Bottom-Up Integration** and **Top-Down Integration**
  - Bottom-Up Integration:
    - This testing begins with unit testing, followed by tests of progressively higher-level combinations of units called modules or builds.
    - Usually done before Top-Down Integration
  - Top-Down Integration:
    - In this testing, the highest-level modules are tested first and progressively, lower-level modules are tested thereafter.
    - We start with system tests and work out way down to unit tests
- **System Testing:** (Functional Testing)
  - Tests the system as a whole
  - Once all the components are integrated, the application as a whole is tested rigouroulsy to see that it meets the specified Quality Standards
  - Generally done by a specialized System Testing Team
  - Important because:
    - System testing is the first step in the Software Development Life Cycle, where the application is tested as a whole.
    - The application is tested thoroughly to verify that it meets the functional and technical specifications.
    - The application is tested in an environment that is very close to the production environment where the application will be deployed.
    - System testing enables us to test, verify, and validate both the business requirements as well as the application architecture.

Testing levels *continued*

- **Regression Testing:** (Functional Testing)
  - Whenever a change in the application is made, it is possible that some related area within the application may have been affected by the change
  - Regression Testing is done to ensure that a change in the application hasn't resulted in the failure of another functionality or feature
  - Intent is to ensure that a change (bug fix, software update, etc.) should not result in another fault being introduced to the application
  - Important because:
    - Minimize the gaps in testing when an application with changes made has to be tested.
    - Testing the new changes to verify that the changes made did not affect any other area of the application.
    - Mitigates risks when regression testing is performed on the application.
    - Test coverage is increased without compromising timelines.
    - Increase speed to market the product.
- **Acceptance Testing:** (Functional Testing)
  - Most important type of testing
  - Done by the QA team who will determine whether the application meets the requirements or not
  - QA team will have a set of pre-written test scenarios and test cases that will be used to test the application
  - By performing Acceptance Testing, the QA Team will determine how the application will perform in Production

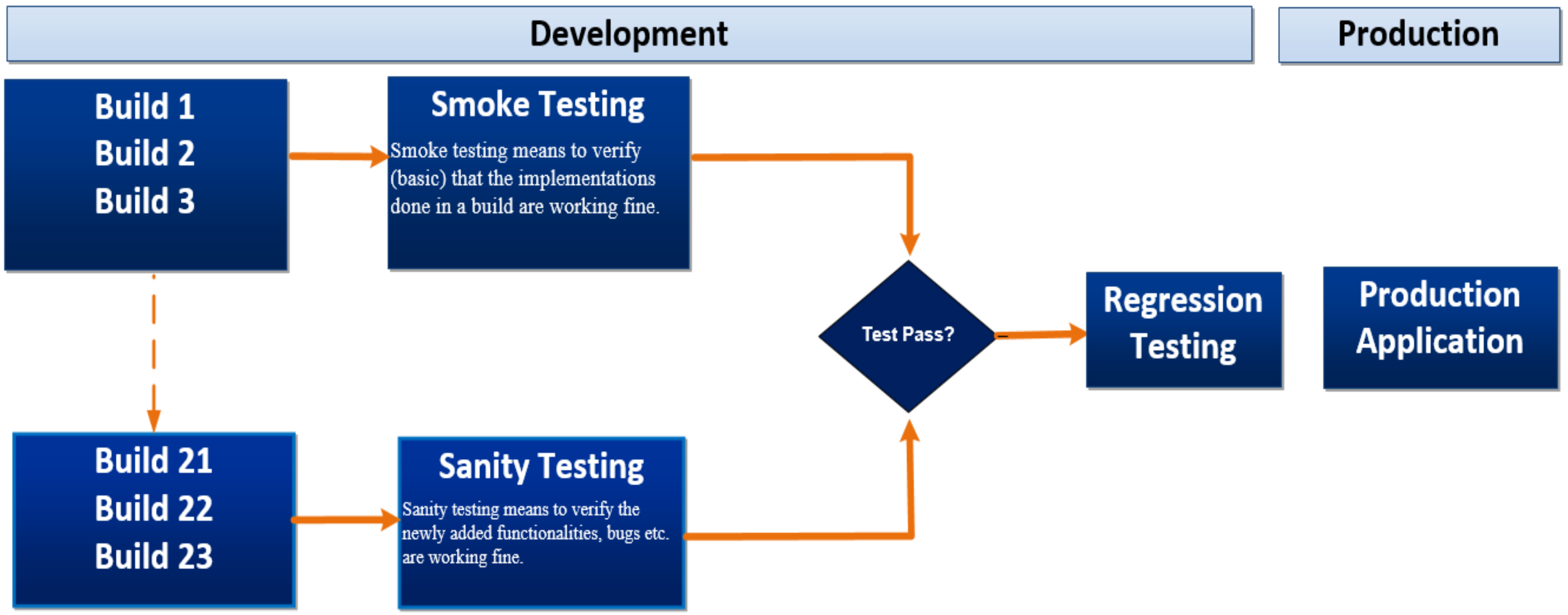
Testing levels *continued*

- **Alpha Testing**: (Functional Testing)
  - This test is the first stage of testing and will be performed amongst the teams (developer and QA teams)
  - Unit testing, integration testing and system testing when combined together is known as **alpha testing**
  - During this phase, the following aspects will be tested in the application:
    - Spelling Mistakes
    - Broken Links
    - Cloudy Directions
    - The Application will be tested on machines with the lowest specification to test loading times and any latency problems.
- **Beta Testing**: (Functional Testing)
  - This test is performed after alpha testing has been successfully performed.
  - A sample of the intended audience tests the application
  - Also known as **pre-release testing**
  - The testers will be testing the following:
    - Users will install, run the application and send their feedback to the project team.
    - Typographical errors, confusing application flow, and even crashes.
    - Getting the feedback, the project team can fix the problems before releasing the software to the actual users.
    - The more issues you fix that solve real user problems, the higher the quality of your application will be.
    - Having a higher-quality application when you release it to the general public will increase customer satisfaction.

Testing levels *continued*

- **Non-Functional Testing**:
  - Involves testing a software against the requirements which are nonfunctional in nature but important such as performance, security, user interface, etc.
  - Types of Non-Functional Testing:
    - <u>Performance Testing</u>:
      - Used to identify any bottlenecks or performance issues rather than finding bugs in a software
      - Causes that contribute in lowering the performance of a software:
        - Network delay
        - Client-side processing
        - Database transaction processing
        - Load balancing between servers
        - Data rendering
      - We test various parameters including:
        - Speed (i.e. Response Time, data rendering and accessing)
        - Capacity
        - Stability
        - Scalability

Testing levels *continued*


Development
Production
Build 1
Build 2
Build 3
Smoke Testing
Smoke testing means to verify (basic) that the implementations done in a build are working fine.
Test Pass?
Regression Testing
Production Application
Build 21
Build 22
Build 23
Sanity Testing
Sanity testing means to verify the newly added functionalities, bugs etc. are working fine.

Testing levels *continued*

- **Load Testing: (Non-Functional Testing)**
  - Process of testing the behavior of a software by applying maximum load in terms of software accessing and manipulating large input data
  - This type of testing identifies the maximum capacity of software and its behavior at peak time.
  - Must be performed with the use of Testing tools to ensure accurate results
  - Virtual users (VUsers) are defined in the automated testing tool and the script is executed to verify the load testing for the software.
  - The number of users can be increased or decreased concurrently or incrementally based upon the requirements.
- **Stress Testing: (Non-Functional Testing)**
  - Involves testing the behavior of a software under abnormal conditions
  - Example – it may include taking away some resources or applying a load beyond the actual load limit
  - The aim of stress testing is to test the software by applying the load to the system and taking over the resources used by the software to identify the breaking point
  - Performed by testing different scenarios such as:
    - Shutdown or restart of network ports randomly
    - Turning the database on or off
    - Running different processes that consume resources such as CPU, memory, server, etc.

Test Plan

The test plan is a term and a deliverable. The test plan is a document that lists all the activities in a QA project, schedules them, defines the scope of the project, roles & responsibilities, risks, entry & exit criteria, test objective.

Test Plan includes:

- ✓ Introduction to the Test Plan document
- ✓ Assumptions while testing the application
- ✓ List of test cases included in testing the application
- ✓ List of features to be tested
- ✓ What sort of approach to use while testing the software
- ✓ List of deliverables that need to be tested
- ✓ The resources allocated for testing the application
- ✓ Any risks involved during the testing process
- ✓ A schedule of tasks and milestones to be achieved

Test Plan

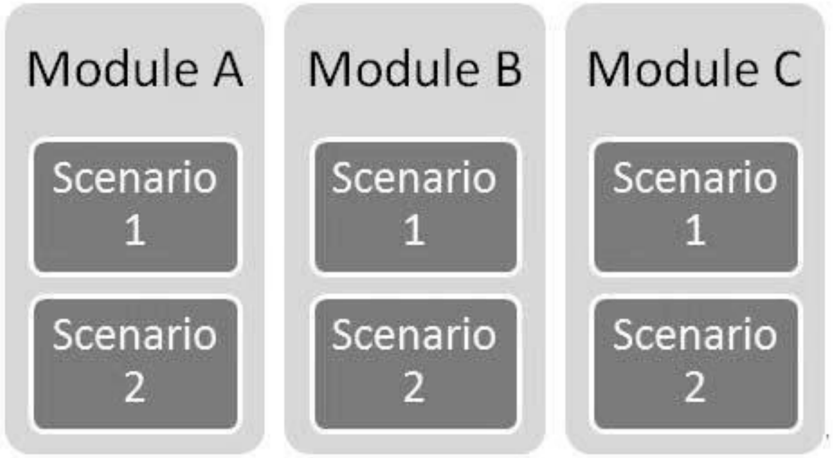
The test plan is a term and a deliverable. The test plan is a document that lists all the activities in a QA project, schedules them, defines the scope of the project, roles & responsibilities, risks, entry & exit criteria, test objective.

Test Plan includes:

- ✓ Introduction to the Test Plan document
- ✓ Assumptions while testing the application
- ✓ List of test cases included in testing the application
- ✓ List of features to be tested
- ✓ What sort of approach to use while testing the software
- ✓ List of deliverables that need to be tested
- ✓ The resources allocated for testing the application
- ✓ Any risks involved during the testing process
- ✓ A schedule of tasks and milestones to be achieved

Test Scenario

- It is a **one line statement** that notifies what area in the application will be tested
- Test scenarios are used to ensure that all process flows are tested from end to end
- A particular area of an application can have as little as one test scenario to a few hundred scenarios depending on the magnitude and complexity of the application
- Each Test Scenario will contain many test cases



Test Scenario

What is a difference between Test scenario and Test condition?

- **Examples test scenarios:**
- 1. Validate if a new country can be added by the Admin
  2. Validate if an existing country can be deleted by the admin
  3. Validate if an existing country can be updated
- **Example test condition:**

  1. Enter the country name as “Canada”(valid)and check for the addition of the country
  2. Enter a blank and check if the country gets added.
  In each case, the specific data is described and the goal of the test is much more precise.

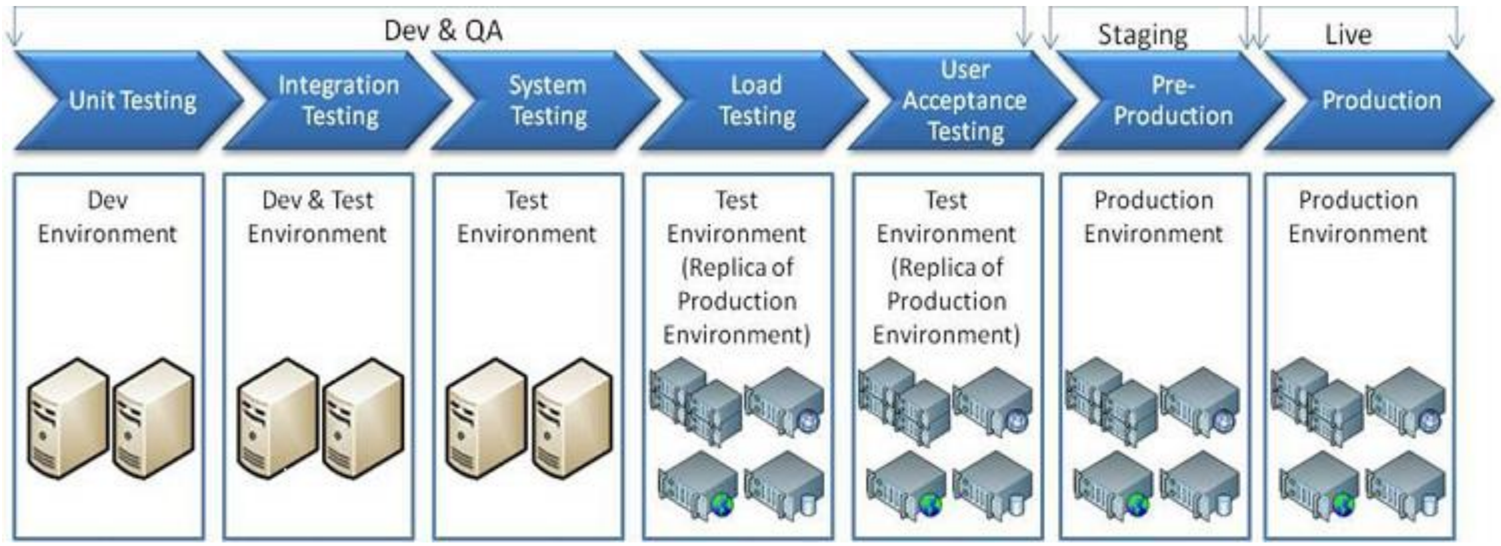
Test Case

- Test cases involve a set of steps, conditions, and inputs that can be used while performing testing tasks
- There are many types of test cases such as functional, negative, error, logical test cases, physical test cases, UI test cases, etc.
- Generally, there are no formal templates that can be used during test case writing
- However, the following components are always available and included in every test case:
  - Test case ID
  - Product module
  - Product version
  - Revision history
  - Purpose
  - Assumptions
  - Pre-conditions
  - Steps
  - Expected outcome
  - Actual outcome
  - Post-conditions
- Many test cases can be derived from a single test scenario
- Sometimes multiple test cases are written for a single software which are collectively known as **test suites**.

Traceability Matrix

- Traceability Matrix (also known as **Requirement Traceability Matrix** - RTM) is a **table** that is used to trace the requirements during the Software Development Life Cycle
- Each requirement in the RTM document is linked with its associated test case so that testing can be done as per the mentioned requirements
- The main goals for this matrix are:
  - Make sure the software is developed as per the mentioned requirements.
  - Helps in finding the root cause of any bug.
  - Helps in tracing the developed documents during different phases of SDLC.

Software Testing Environment



For additional questions or comments please send an email to:
admin@digitalpoint.tech