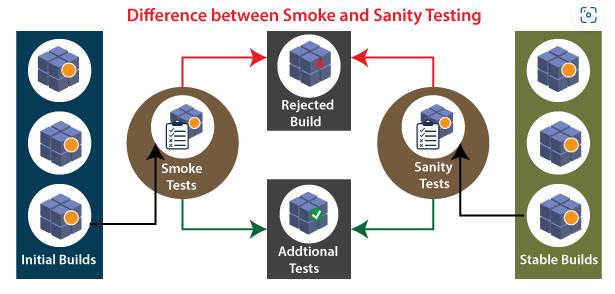
***Q1 ->***

***Smoke and Sanity Testing***

**Smoke Testing**: Smoke testing is a testing technique that is used to check the basic functionality of a software application or system after a build or release. This testing is done to ensure that the build is stable enough for further testing. Smoke testing involves a quick and shallow check of the software application to verify that it is functioning properly and that there are no critical defects that could prevent further testing. Smoke testing is typically performed by testers or developers before any detailed testing is performed.

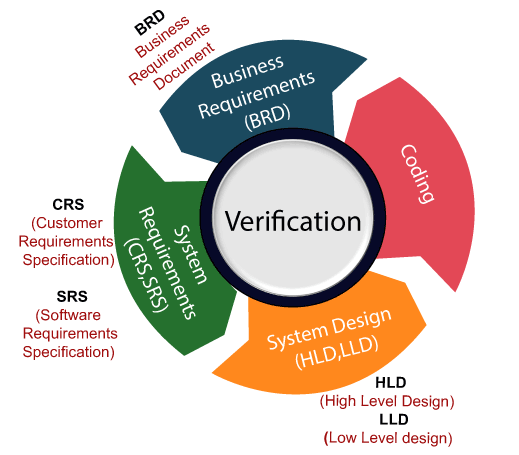
**Sanity Testing**: Sanity testing is a testing technique that is used to check that specific functionality or components of a software application are working as expected after making changes or fixing defects. The main objective of sanity testing is to verify that the changes made to the application have not introduced new defects or issues in the specific functionality or components. Sanity testing is typically performed after regression testing and is focused on specific areas of the application.

|  |  |
| --- | --- |
| **Smoke Testing** | **Sanity Testing** |
| Smoke Testing is performed to ascertain that the critical functionalities of the program is working fine | Sanity Testing is done to check the new functionality/bugs have been fixed |
| The objective of this testing is to verify the “stability” of the system in order to proceed with more rigorous testing | The objective of the testing is to verify the “rationality” of the system in order to proceed with more rigorous testing |
| This testing is performed by the developers or testers | Sanity testing in software testing is usually performed by testers |
| Smoke testing is usually documented or scripted | Sanity testing is usually not documented and is unscripted |
| Smoke testing is a subset of Acceptance testing | Sanity testing is a subset of Regression Testing |
| Smoke testing exercises the entire system from end to end | Sanity testing exercises only the particular component of the entire system |
| Smoke testing is like General Health Check Up | Sanity Testing is like specialized health check up |

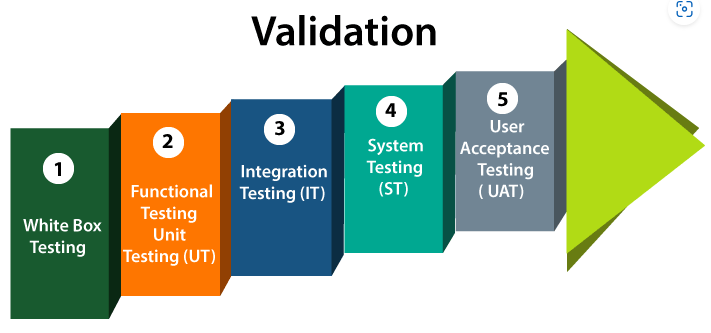


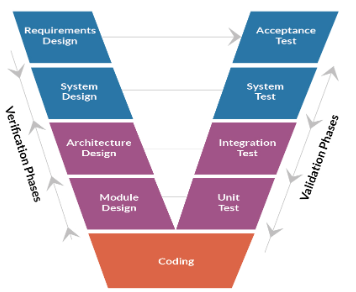
***Verification and Validation Testing***

**Verification testing:** Verification testing includes different activities such as business requirements, system requirements, design review, and code walkthrough while developing a product. It is also known as static testing, where we are ensuring that "we are developing the right product or not". And it also checks that the developed application fulfilling all the requirements given by the client.



**Validation testing:** Validation testing is testing where tester performed functional and non-functional testing. Here functional testing includes Unit Testing (UT), Integration Testing (IT) and System Testing (ST), and non-functional testing includes User acceptance testing (UAT). Validation testing is also known as dynamic testing, where we are ensuring that "we have developed the product right." And it also checks that the software meets the business needs of the client.



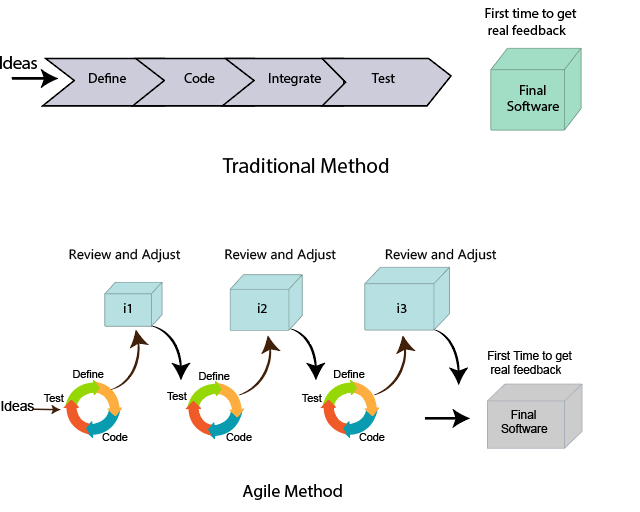


|  |  |
| --- | --- |
| **Verification** | **Validation** |
| We check whether we are developing the right product or not. | We check whether the developed product is right. |
| Verification is also known as static testing. | Validation is also known as dynamic testing. |
| Verification includes different methods like Inspections, Reviews, and Walkthroughs. | Validation includes testing like [functional testing](https://www.javatpoint.com/functional-testing), system testing, [integration](https://www.javatpoint.com/integration-testing), and User acceptance testing. |
| It is a process of checking the work-products (not the final product) of a development cycle to decide whether the product meets the specified requirements. | It is a process of checking the software during or at the end of the development cycle to decide whether the software follow the specified business requirements. |
| Quality assurance comes under verification testing. | Quality control comes under validation testing. |
| The execution of code does not happen in the verification testing. | In validation testing, the execution of code happens. |
| In verification testing, we can find the bugs early in the development phase of the product. | In the validation testing, we can find those bugs, which are not caught in the verification process. |
| Verification testing is executed by the Quality assurance team to make sure that the product is developed according to customers' requirements. | Validation testing is executed by the testing team to test the application. |
| Verification is done before the validation testing. | After verification testing, validation testing takes place. |
| In this type of testing, we can verify that the inputs follow the outputs or not. | In this type of testing, we can validate that the user accepts the product or not. |

***Q2 -> What is Agile Methodology?***

Agile Methodology meaning a practice that promotes continuous iteration of development and testing throughout the software development lifecycle of the project. In the Agile model in software testing, both development and testing activities are concurrent, unlike the Waterfall model.

An agile methodology is an iterative approach to software development. Each iteration of agile methodology takes a short time interval of 1 to 4 weeks. The agile development process is aligned to deliver the changing business requirement. It distributes the software with faster and fewer changes. The single-phase software development takes 6 to 18 months. In single-phase development, all the requirement gathering and risks management factors are predicted initially. The agile software development process frequently takes the feedback of workable product. The workable product is delivered within 1 to 4 weeks of iteration.





***Q3 -> Epic and User Stories***

**What’s a User Story?**

A user story is a short, written description of how the software should work from the perspective of the end user. A good user story will include an action or event, an actor (the person or thing that makes the request, or the user), and a result. The result can be either in direct response to the action or event of the user story, or it can be something that follows the action.

For example:

* As a customer, I want to know about upcoming events so I can plan my visit.
* As a visitor, I want to see upcoming events on my mobile device.

**What’s an Epic in Agile?**

An epic is probably too big to fit into a sprint and needs to be broken down into stories and tasks. Epics are usually defined during the initial product roadmap or backlog and broken down into stories as further knowledge is gained in the product list. Epics are written in a user story format in story mapping. The stories in an epic have a common purpose and a specific outcome, a high-level user need, or a part of a journey or process taken while using the product.

In agile software development, an epic is a lengthy document that provides a summary of the features of your product. Epics are good for communicating the scope of your product to stakeholders and will help you get everyone on the same page when it comes to what should be done.

Epics are typically one or two pages long and should have a detailed description of the feature as well as its benefits for the user. This includes a snapshot of how this feature fits into your product roadmap, what’s needed from other teams to complete this feature, the timeline for when this feature will be completed, and how this will affect your customers.

This process allows you to break down your project into smaller parts so it’s easier to manage and work on. It also ensures everyone is on the same page when it comes to what they need to do next. Other pros of epics are that they help keep track of all the different features in your product, as well as give team members an idea of what needs to be done next.

