***Unit Testing*** is the method of verifying the smallest piece of testable code against its purpose.” If the purpose or requirement fails then the unit test has failed.

In simple words, it means – writing a piece of code (unit test) to verify the code (unit) written for implementing requirements.

For Unit testing, developers use manual or automated tests to ensure that each unit in the software meets the customer’s requirement. This unit can be an individual function, object, method, procedure, or module in the software under test.

Writing unit tests to test the individual units makes writing comprehensive tests easier as all the units are put together. During software development, it is done as the first level of testing.

Unit Testing is used to design robust software components that help maintain code and eliminate issues in code units. We all know the importance of finding and fixing defects in the early stages of the software development cycle. This testing serves the same purpose.

**Unit Testing Methods**

**It can be performed in 2 ways :**

1. Manual Testing
2. Automated Testing

In **Manual Testing**, the tester manually executes test cases without using any automation tool. Here, each stage of the test is executed manually. Manual Testing is tedious especially for tests that are repetitive and requires more effort to create and execute test cases. Manual Testing does not require knowledge of any testing tool.

It is a fact that 100% of Automation is not possible and thus there will always be some level of manual testing performed.

In**Automated Testing,** software testing automation tools are used to automate test/test cases. The automation tool can record and save your test and it can be re-played as many times as needed without any further human intervention.

These tools can even enter test data into the system being tested as well as it can compare the expected results to the actual results and automatically generate the reports. However, the initial cost of setting up test automation tools is high.

**Benefits of Unit Testing**

1. **The process becomes agile:** To add new functions or features to the existing software, we need to make changes to the old code. But changing things to the already tested code can be risky as well as costly.
2. **Code quality improves:** The quality of the code is automatically improved when unit testing is done. The bugs identified during this testing are fixed before they are sent for the integration testing phase. Results in robust design and development as developers write test cases by understanding the specifications first.
3. **Detects bugs early:** As developers run unit tests, they detect bugs early in the software development life cycle and resolve them. This includes flaws or missing parts in the specification as well as bugs in the programmer’s implementation.
4. **Easier changes and simplified integrations:** Doing unit testing makes it easy for the developer to restructure the code, make changes, and maintain the code. It also makes testing the code after integration much easier. Fixing an issue in Unit Testing can fix many other issues occurring in later development and testing stages
5. **Documentation availability:** Developers who are looking into the functionality at a later stage can refer to the unit testing documentation and can easily find the unit test interface and correct or work fast and easily.
6. **Easy debugging process:** It helps in simplifying the debugging process. If the test fails at any stage the code needs to be debugged or else the process can be continued without any obstacles.
7. **Lower cost:** When bugs are detected and resolved during unit testing, cost and development time is reduced. Without this testing, if the same bugs are detected at a later stage after the code integration, it becomes more difficult to trace and resolve, making it more costly and increasing development time.
8. **Code completeness can be demonstrated using unit tests:** This is more useful in the agile process. Testers don’t get functional builds to test until integration is completed. Code completion cannot be justified by showing that you have written and checked the code. However, running Unit tests can demonstrate code completeness.
9. **Save development time:** Code completion may take longer but due to fewer bugs in the System and Acceptance testing, overall development time can be saved.
10. [Code coverage](https://www.softwaretestinghelp.com/test-coverage/) can be measured.

**How to write good Unit Tests?**

* Unit tests should be written to verify a single unit of code and not the integration.
* Small and isolated Unit tests with clear naming would make it very easy to write and maintain.
* Changing to another part of the software should not affect the Unit test if those are isolated and written for a specific unit of code.
* It should run quickly.
* Unit test should be reusable.

## **Conclusion**

Unit testing comes into picture when it is required to test every function separately. It is much more reasonable to detect and fix bugs during this testing and save time & cost, rather than finding them at the later stage of software development.

While it offers many advantages, there are also limitations involved with using it. **Rigorous discipline and consistency are required throughout the software development process** to overcome limitations and get the intended benefits.

**Integration testing.**

The meaning of Integration testing is quite straightforward- Integrate/combine the unit tested module one by one and test the behavior as a combined unit.

The main function or goal of this testing is to test the interfaces between the units/modules.

We normally do Integration testing after “Unit testing”. Once all the individual units are created and tested, we start combining those “Unit Tested” modules and start doing the integrated testing.

The main function or goal of this testing is to test the interfaces between the units/modules.

The individual modules are first tested in isolation. Once the modules are unit tested, they are integrated one by one, till all the modules are integrated, to check the combinational behavior, and validate whether the requirements are implemented correctly or not.

Here we should understand that Integration testing does not happen at the end of the cycle, rather it is conducted simultaneously with the development. So in most of the times, all the modules are not actually available to test and here is what the challenge comes to test something which does not exist!

**Why Integration Test?**

We feel that Integration testing is complex and requires some development and logical skill. That’s true! Then what is the purpose of integrating this testing into our testing strategy?

**Here are some reasons:**

1. In the real world, when applications are developed, it is broken down into smaller modules and individual developers are assigned 1 module. The logic implemented by one developer is quite different than another developer, so it becomes important to check whether the logic implemented by a developer is as per the expectations and rendering the correct value in accordance with the prescribed standards.
2. Many a time the face or the structure of data changes when it travels from one module to another. Some values are appended or removed, which causes issues in the later modules.
3. Modules also interact with some third party tools or APIs which also need to be tested that the data accepted by that API / tool is correct and that the response generated is also as expected.
4. A very common problem in testing – Frequent requirement change! :) Many a time developer deploys the changes without unit testing it. Integration testing becomes important at that time.

**Advantages**

* This testing makes sure that the integrated modules/components work properly.
* Integration testing can be started once the modules to be tested are available. It does not require the other module to be completed for testing to be done, as Stubs and Drivers can be used for the same.
* It detects the errors related to the interface.

### Challenges

**#1)** Integration testing means testing two or more integrated systems in order to ensure that the system works properly. Not only the integration links should be tested but an exhaustive testing considering the environment should be done to ensure that the integrated system works properly.

There might be different paths and permutations which can be applied to test the integrated system.

**#2)** Managing Integration testing becomes complex because of few factors involved in it like the database, Platform, environment etc.

**#3)** While integrating any new system with the legacy system, it requires a lot of changes and testing efforts. Same applies while integrating any two legacy systems.

**#4)** Integrating two different systems developed by two different companies is a big challenge as for how one of the systems will impact the other system if any changes are done in any one of the systems is not sure.

In order to minimize the impact while developing a system, few things should be taken into consideration like possible integration with other systems, etc.

### Types of Integration Testing

Big bang approach integrates all the modules in one go i.e. it does not go for integrating the modules one by one. It verifies if the system works as expected or not once integrated. If any issue is detected in the completely integrated module, then it becomes difficult to find out which module has caused the issue.

Big bang approach is a time-consuming process of finding a module which has a defect itself as that would take time and once the defect is detected, fixing the same would cost high as the defect is detected at the later stage.

Bottom-up testing, as the name suggests starts from the lowest or the innermost unit of the application, and gradually moves up. The Integration testing starts from the lowest module and gradually progresses towards the upper modules of the application. This integration continues till all the modules are integrated and the entire application is tested as a single unit.

#### Top-down approach

This technique starts from the topmost module and gradually progress towards the lower modules. Only the top module is unit tested in isolation. After this, the lower modules are integrated one by one. The process is repeated until all the modules are integrated and tested.

### **GUI application Integration Test**

### Graphical User-interface Testing or GUI testing is a process of testing the user interface of an application. In this article, you will learn about the basics of GUI testing and how to do GUI testing and its benefits.

Integration test cases focus mainly on the**interface between the modules, integrated links, data transfer** between the modules as modules/components that are already unit tested i.e. the functionality and the other testing aspects have already been covered.

Integration testing technique can be counted in both black boxes as well as [white box technique](https://www.softwaretestinghelp.com/white-box-testing-techniques-with-example/). [Black box technique](https://www.softwaretestinghelp.com/black-box-testing/) is where a tester need not have any internal knowledge of the system i.e. coding knowledge is not required whereas white box technique needs internal knowledge of the application.

Now while performing integration testing it could include testing the two integrated web services which will fetch the data from the database & provide the data as required which means it could be tested using white box testing technique whereas integrating a new feature in the website can be tested using the black box technique.

So, it’s not specific that integration testing is a black box or white box technique.

**Integration Testing Tools**

* Rational Integration Tester
* Protractor
* Steam
* TESSY

### Conclusion

Test Integration is an important part of the testing cycle as it makes it easier to find the defect when two or more modules are integrated in order to integrate all the modules all together in the first step itself.

It helps in finding the defects at an early stage which in turn saves the effort and cost as well. It ensures that the integrated modules work properly as expected.

**System Testing** means testing the system as a whole. All the modules/components are integrated in order to verify if the system works as expected or not.

System Testing is done after Integration Testing. This plays an important role in delivering a high-quality product.

The process of testing an integrated hardware and software system to verify that the system meets its specified requirements.

**Verification**: Confirmation by examination and provisions of objective evidence that specified requirements have been fulfilled.

If an application has three modules A, B, and C, then testing done by combining the modules A & B or module B & C or module A& C is known as Integration testing. Integrating all the three modules and testing it as a complete system is termed as System testing.

It is mainly a Black-box type testing. This testing evaluates the working of the system from a user point of view, with the help of a specification document. It does not require any internal knowledge of systems like the design or structure of the code.

It contains functional and non-functional areas of application/product.

**It mainly focuses on the following:**

1. External interfaces
2. Multiprogram and complex functionalities
3. Security
4. Recovery
5. Performance
6. Operator and user’s smooth interaction with the system
7. Install ability
8. Documentation
9. Usability
10. Load/Stress

**Why System Testing?**

**#1)** It is very important to complete a full test cycle and ST is the stage where it is done.

**#2)** ST is performed in an environment that is similar to the production environment and hence stakeholders can get a good idea of the user’s reaction.

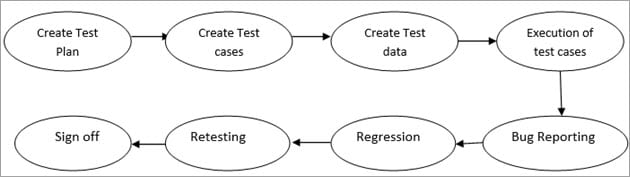
**#3)** It helps to minimize after-deployment troubleshooting and support calls.

**#4**) In this STLC stage Application Architecture and Business requirements, both are tested.

System testing can be considered as a black-box test technique.

[Black box Testing](https://www.softwaretestinghelp.com/black-box-testing/) technique does not require internal knowledge of the code whereas the white box technique requires internal knowledge of the code.

While performing System testing functional & non-functional, security, Performance and many other testing types are covered and they are tested using a black-box technique wherein the input is provided to the system and the output is verified. System internal knowledge is not required.



### Conclusion

System testing is very important and if not done properly critical issues can be faced in the live environment.

A system as a whole has different characteristics to be verified. A simple example would be any website. If it’s not tested as a whole then the user might find that site to be very slow or the site might get crashed once a large number of users log in at the same time.

And these characteristics cannot be tested until the website is tested as a whole.

***User Acceptance Testing (UAT), also known as beta or end-user testing, is defined as testing the software by the user or client to determine whether it can be accepted or not.****This is the final testing performed once the functional, system and regression testing are completed.*

*The main purpose of this testing is to validate the software against the business requirements. This validation is carried out by the end-users who are familiar with the business requirements.*

*UAT,*[*alpha and beta testing*](https://www.softwaretestinghelp.com/what-is-alpha-testing-beta-testing/)*are different types of acceptance testing.*

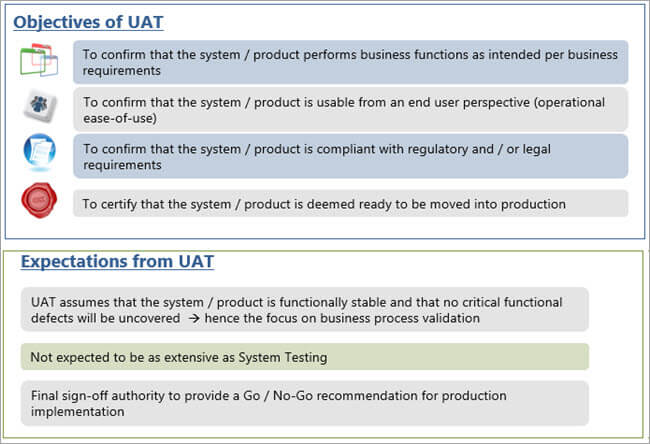
As the user acceptance test is the last testing that is carried out before the software goes live, obviously this is the last chance for the customer to test the software and measure if it is fit for the purpose.

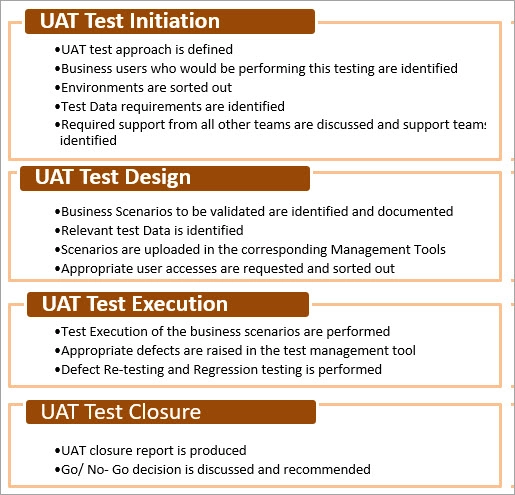
This is typically the last step before the product goes live or before the delivery of the product is accepted. This is performed after the product itself is thoroughly tested

### Who Performs UAT?

Users or client – This could be either someone who is buying a product (in the case of commercial software) or someone who has had a software custom-built through a software service provider or the end-user if the software is made available to them ahead of the time and when their feedback is sought out.

The team can be comprised of beta testers or the customer should select UAT members internally from every group of the organization so that each and every user role can be tested accordingly.





Usually, UAT is undertaken by a Subject Matter Expert (SME) and /or a business user, who might be the owner or the customer of a system under test. Similar to the System testing phase, the UAT phase also encompasses religious phases before it is brought to closure.

### Tools & Methodologies

Typically, the type of software tools that are used during this testing phase is similar to the tools used while performing functional testing.

As this phase involves validating the complete end to end flows of the application, it might be difficult to have one tool to automate this validation completely. However, to some extent, we would be able to leverage the automated scripts developed during system testing.

Similar to system testing, users would also use test management and defect management tool like QC, JIRA, etc. These tools can be configured to cumulate data for the User Acceptance phase.

### UAT In Agile Environment

The agile environment is more dynamic in nature. In an agile world, business users will be involved throughout the project sprints and the project would be enhanced based on the feedback loops from them.

At the beginning of the project, business users would be the key stakeholders to provide requirement thereby updating the product backlog. During the end of each sprint, business users would participate in the sprint demo and would be available for providing any feedback.

Moreover, a UAT phase would be planned before the sprint completion where the business users would do their validations.

The feedbacks which are received during sprint demo and sprint UAT, are collated and added back to the product backlog which is constantly reviewed and prioritized. Thus in an agile world, the business users are more close to the project and they evaluate the same for its use more frequently unlike the traditional waterfall projects.

## **Conclusion**

**#1)**UAT is not about the pages, fields or buttons. The underlying **assumption** even before this test begins is that all that basic stuff is tested and is working fine. God forbid, the users find a bug as basic as that – it is a piece of very bad news for the QA team. :(

**#2)** This testing is about the entity that is the primary element in the business.

**Let me give you an example:** If the AUT is a ticketing system, the UAT is not going to be about, searching for the menu that opens a page, etc. It is about the tickets and their reservation, the states that it can take, its journey through the system, etc.

Another **Example,** if the site is a car dealership site, then the focus is on the “car and its sales” and not really the site. Hence, the core business is what is verified and validated and who is better to do it than the business owners. That’s why this testing makes the most sense when the customer is involved to a major extent.

**#3)** UAT is also a form of testing at its core which means **that there is a good chance of identifying some bugs at this phase too**. It sometimes happens. Aside from the fact that it is a major escalation on the QA team, the UAT bugs usually mean a meeting to sit and discuss how to handle them as following this testing there is usually no time to fix and retest.

**The decision would be either to:**

* Push the go-live date, fix the issue first and then move on.
* Leave the bug as it is.
* Consider it as a part of the change request for future releases.

**#4)**UAT is classified as Alpha and Beta testing, but that classification is not that important in the context of typical software development projects in a service-based industry.

* **Alpha testing** is when UAT is carried out in the software builder’s environment and is more significant in the context of commercial off the shelf software.
* [**Beta testing**](https://www.softwaretestinghelp.com/what-is-alpha-testing-beta-testing/) is when the UAT is carried out in the production environment or the client’s environment. This is more common for customer-facing applications. The users here are the actual customers like you and me in this context.

**#5)** Most of the time in a regular software development project, UAT is carried out in the [QA environment](https://www.softwaretestinghelp.com/test-bed-test-environment-management-best-practices/) if there is no staging or UAT environment.

**In short,** the best way to find out if your product is acceptable and fit for purpose is to actually put it in front of the users.

Organizations are getting into the Agile way of delivering, business users are getting more involved and the projects are being enhanced and delivered via feedback loops. All being done, the User Acceptance phase is considered as the gate for getting into implementation and production.