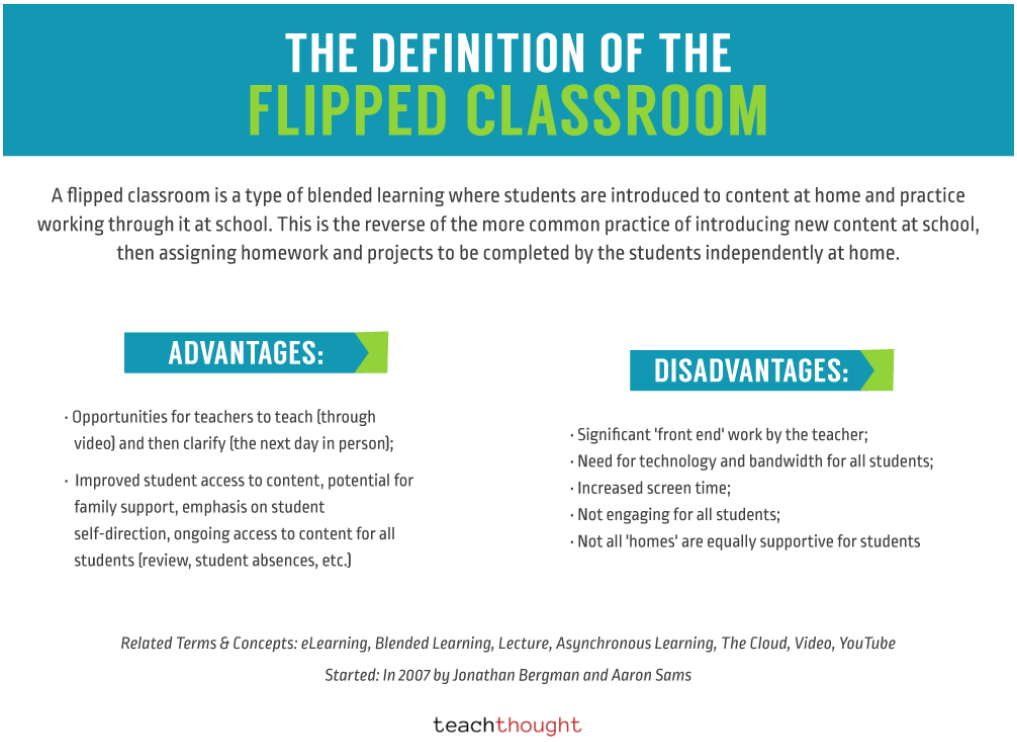
INNOVATION ON FLIPPED LEARNING PROCESS

Flipped learning

Flipped learning is a methodology that helps teachers to prioritize active learning during class time by assigning students lecture materials and presentations to be viewed at home or outside of class. One of the most exciting advancements in the modern classroom is flipped learning.

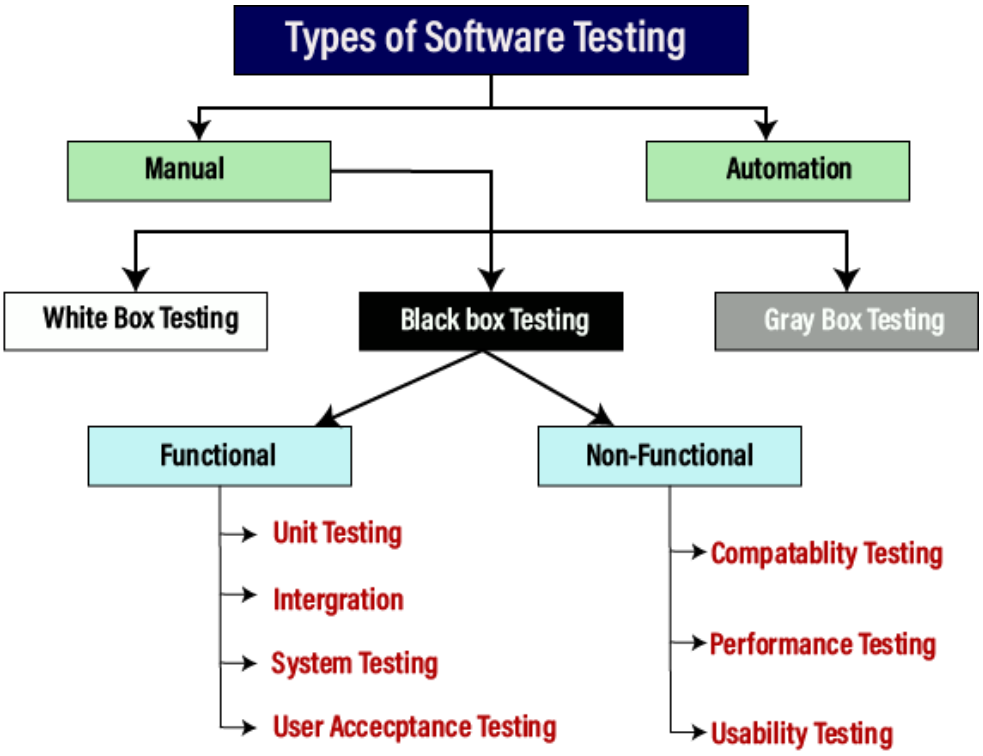


FLIPPED LEARNING PROCESS IN

SOFTWARE TESTING

SOFTWARE TESTING

Software Testing is a method to check whether the actual software product matches expected requirements and to ensure that software product is Defect free. It involves execution of software/system components using manual or automated tools to evaluate one or more properties of interest.

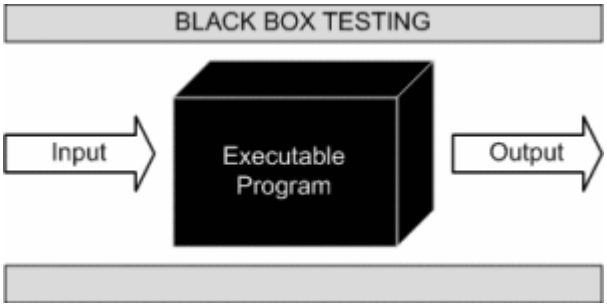


White Box Testing

White box testing is an approach that allows testers to inspect and verify the inner workings of a software system—its code, infrastructure, and integrations.

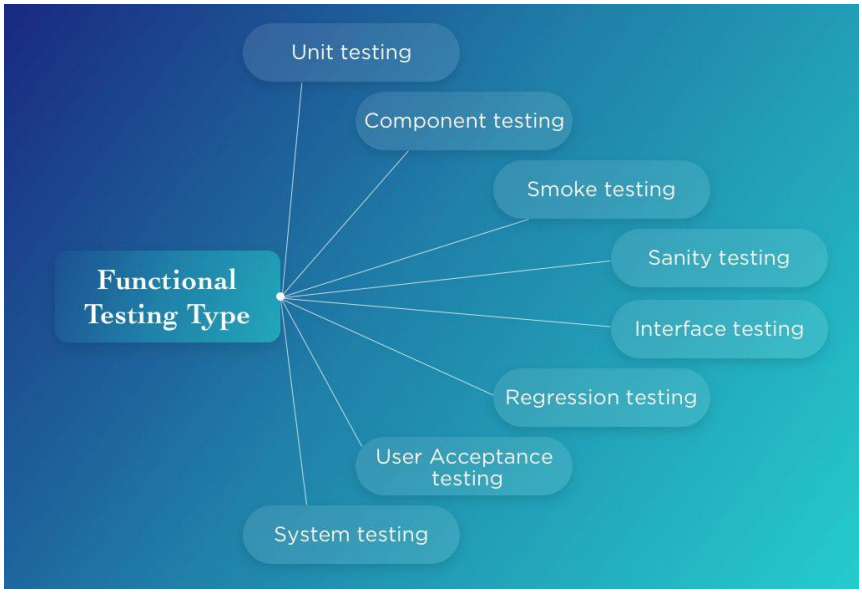
Black -box testing

Black box testing is a technique of software testing which examines the functionality of software without peering into its internal structure or coding.



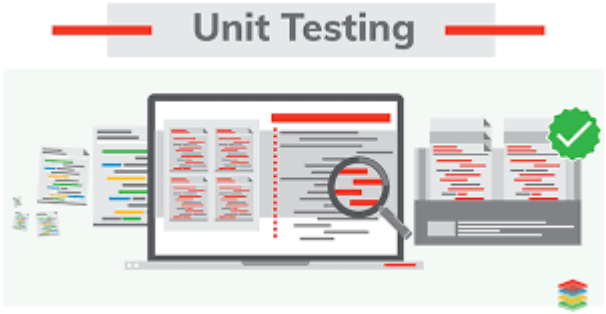
Functional Testing

Functional testing is the process through which QAs determine if a piece of software is acting in accordance with pre-determined requirements.



INTRODUCTION

Unit testing



Unit tests are typically automated tests written and run by software developers to ensure that a section of an application (known as the "unit") meets its design and behaves as intended. In procedural programming, a unit could be an entire module, but it is more commonly an individual function or procedure. In object-oriented programming, a unit is often an entire interface, such as a class, or an individual method. By writing tests first for the smallest testable units, then the compound behaviors between those, one can build up comprehensive tests for complex applications.

To isolate issues that may arise, each test case should be tested independently. Substitutes such as method stubs, mock objects, fakes, and test harnesses can be used to assist testing a module in isolation.

During development, a software developer may code criteria, or results that are known to be good, into the test to verify the unit's correctness. During test case execution, frameworks log tests that fail any criterion and report them in a summary. For this, the most commonly used approach is test - function - expected value.

Writing and maintaining unit tests can be made faster by using parameterized tests. These allow the execution of one test multiple times with different input sets, thus reducing test code duplication. Unlike traditional unit tests, which are usually closed methods and test invariant conditions, parameterized tests take any set of parameters. Parameterized tests are supported by TestNG, JUnit and its .Net counterpart, XUnit. Suitable parameters for the unit tests may be supplied manually or in some cases are automatically generated by the test framework. In recent years support was added for writing more powerful (unit) tests, leveraging the concept of theories, test cases that execute the same steps, but using test data generated at runtime, unlike regular parameterized tests that use the same execution steps with input sets that are pre-defined

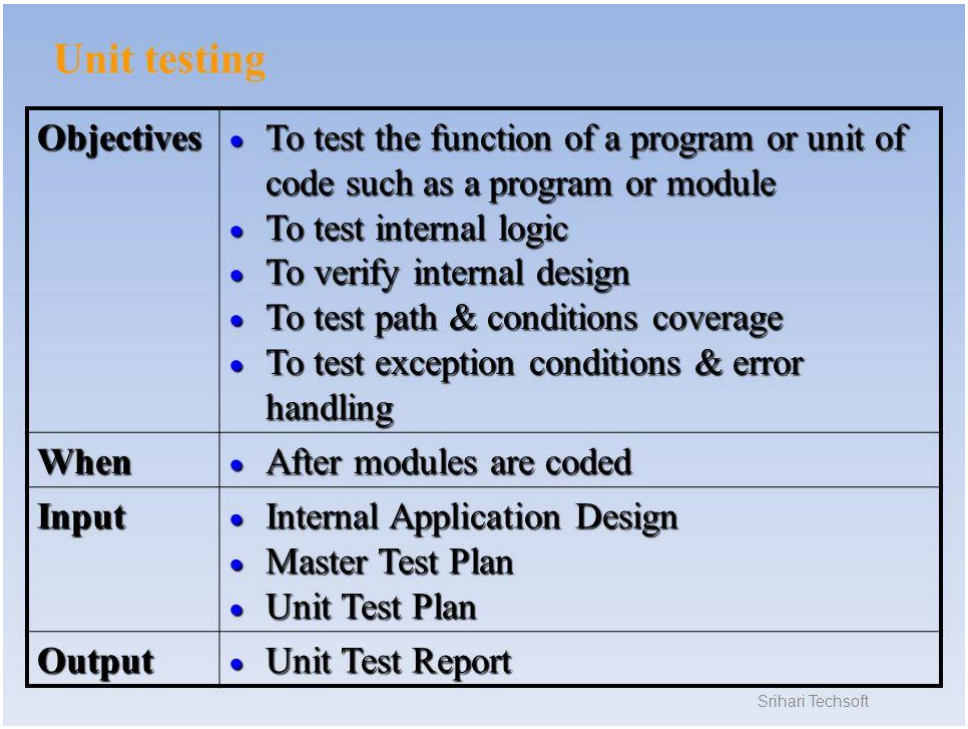
OBJECTIVE OF THE ACTIVITY

UNIT TESTING

The main objective of unit testing is to ensure that each individual part is working well and as it's supposed to work. The entire system will only be able to work well if the individual parts are working well. Unit testing is performed by the software developers themselves. Sometimes, independent software testers also perform these tests.

There are two main types of unit testing: manual and automated. The automated method is the most preferred as it is faster and more accurate, but performing this task manually is also an option. The manual approach has a step by step instructional procedure that helps testers perform this task efficiently. The automated unit testing usually involves the developer first writing a section of the code in the application so that the function can be tested.

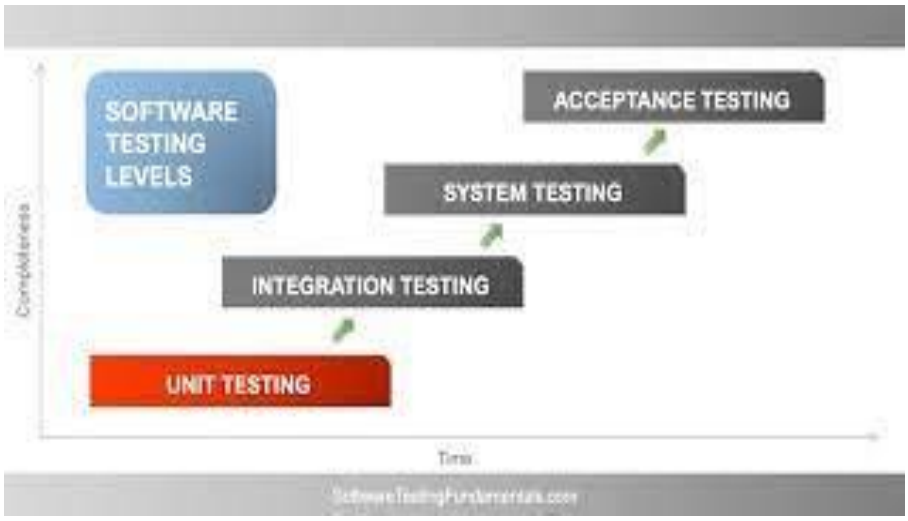
After that, when the application is deployed, they remove the test code. They can also isolate the function to test it in a more thorough way. This helps with identifying any dependency that might be there between the tested code and the other data spaces. These dependencies can then be eliminated.



EXECUTION PLAN

Steps for execution

1. Analyze the product or feature you're testing.
2. Design the test strategies (and approach) you're going to use.
3. Define the test objectives and pass/fail criteria.
4. Plan the test environment.
5. Execute your test plan and track progress in your project management tool.



Designing the Test Plan

The purpose of creating a test plan is to have a documented way to execute consistent, measurable testing on an enterprise wide basis. A well designed test plan will address the following concerns:

- What is to be tested?
- When is testing to be conducted?
- Who or what will do the testing?
- How will test results be stored?
- How are test results to be evaluated as successful?
- How will test results be reported?
- How will tests be maintained and enhanced?

What is to be tested?

As described in Table 1 above, the items under test will vary with each phase of the Software Development Deployment Process. Each phase will have it's own scope of testing. For example, in the Development Phase of the SDDP, developers will create and execute unit tests against source code. In the General QA phase, testers and test engineers will test UI of an application, as well as service and API endpoints.

In addition to describing the items to test at each phase in the SDDP, a good test plan will define the standard for adequate testing. For example, in terms of unit testing, a test plan might require that all public functions of a component to be tested and that all the functions tested must pass.

Also, it's typical that a test plan will define the code coverage requirement that unit tests must satisfy. Some shops demand 100% code coverage while other are less strict. Having a well defined itemization of what is to be tested and a documented standard by which to determine adequate, successful testing is a critical part of the test plan, particularly when it come time to report on the results of testing activity.

When is testing to be conducted?

Testing can become a burdensome cost if not managed well. Testing everything all the time doesn't make sense, both in terms of reasonable testing practices and efficient utilization of resources. For example, running a complete test suite upon a code base makes sense when the code base or operational environment has changed. Running a test just because you can is a waste.

Thus, a good test plan will clearly describe when testing is to be conducted. The time of test execution will vary according to the needs of SDDP phase. For example, it's usual for unit tests to be run automatically by the CI/CD deployment tool whenever feature code is merged into a common branch. A failing unit test will stop the merge activity within the CI/CD from continuing.

Typically, the full suite of functional and integration tests are executed when the code base is escalated into the next deployment phase. The time and place of testing activity must be known to all involved in the software development process. Some shops will send emails to interested testing parties when tests are due to be executed. Others will keep the schedule of test events on the company wiki.

The important thing is that communicating testing times must be part of the test plan and must be made to known to all.

Who or what will do the testing?

Part of any test plan must be a clear declaration of who or what is to conduct a given test. In the case of automated testing, the test plan must describe the automation tools and agents that will do the testing. In terms of manual testing, the test plan will describe the group or individuals responsible for creating and executing tests.

Defining who or what will do testing might seem like obvious information. However, it's not, particularly for shops that are trying to make the transition to supporting Test Driven Development. Who or what will do testing must be stated as a matter of policy in the test plan. For example, a policy declaration that all developers are responsible for testing the code they write leaves no ambiguity whatsoever about the relationship of developer to unit testing.

Declaring that all API testing in QA will be conducted using a tool such as LoadUI makes clear the toolset that will do the testing and the mastery required by staff to perform testing. Defining in the test plan who or what will do a particular aspect of testing creates a uniformity to the testing process which reduces costs and creates efficiency.

How will test results be stored?

How and where they will be stored needs to be part of the test plan. Some tests plans will rely upon the storage features of test framework to make sure that test data is saved and retrievable. Other times, saving test results might need to take place via logs or in a database.

As such, the test plan needs to describe the logging or database technology used, where the data will reside and how the data will be accessed. Many testing frameworks will store only the results of the last test suite run. However, some companies need to have a history of test reports in order to identify operational trends.

Should the enterprise need to keep track of testing history, this requirement needs to be accounted for in the test plan. Clearly defining how and where test results will be stored is an important part of any test plan. Otherwise the enterprise runs the risk of losing mission critical data when the people who do know where the information leave the company.

How are test results to be evaluated as successful?

A good test plan will articulate in a clear, quantitative manner how success is to be determined for any testing session in the software deployment process. For example, in terms of unit testing success, the test plan can define a pass/fail and code coverage standard as described earlier. In terms of performance testing, success can be measured by setting the maximum amount of time that can elapse when a given HTTP request executes.

The important thing is that the test plan must describe how success will be determined for any and all tests. Enterprises need to determine success according to a quantitative standard. Otherwise, there is no objectivity in testing. Consistent quality depends on an measurable standard for success.

How will test results be reported?

Test results need to be reported in order to be useful. A test plan needs to describe the information that will be gathered and evaluated for reporting. Also, the plan needs to describe how reporting will be made available. In terms of test data, there's operational test reporting and project level test reporting.

Operational reporting describes the result of a given test and provides developers, test engineers and testers with information that can be used to fix bugs immediately. Project level test reporting is intended for management and project sponsors. Project level test reports contain summaries of test results, historical information, and analysis of test data.

Management and project sponsors use project level test reports to make business decisions relevant to the project and personnel associated with the project. Test results can be reported as part of the build process in a project dashboard. Also, they can be a set of executive summaries delivered to key stakeholders via email. These are but a few of the delivery options.

A good test plan will leave nothing to chance. When designing a test plan, make sure the plan includes a detailed list of reports to be issued, the intended recipients and the means by which reports will be distributed.

How will manual and automated tests be maintained and enhanced?

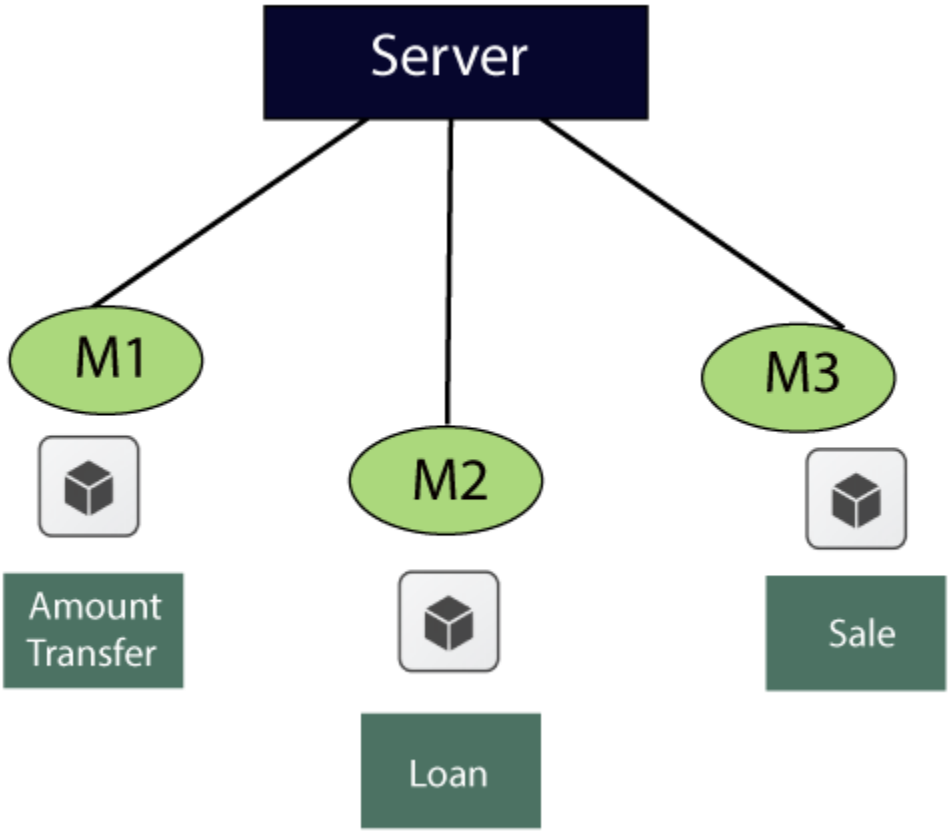
Test plans, like software itself, evolve over time. Thus, tests will need to be updated to keep in step with changes made to the software and in the organization. The test plan needs to describe how tests will be maintained and enhanced. Some tests, such as unit tests, will be part of the code base and can be stored in the source code repositories along with the code base.

Functional tests scripts might be stored in repositories too. Test reports intended for management and project sports contain sensitive information. Thus, these types of reports are best stored in the company's document management system. Storing sensitive information on a file server is risky unless the company has a well defined security protocol for storage on a network drive.

PLAN OF ACTION

Example of Unit testing

Let us see one sample example for a better understanding of the concept of unit testing:



For the **amount transfer,** requirements are as follows:

1.	Amount transfer
1.1	From account number (FAN)→ Text Box
1.1.1	FAN→ accept only 4 digit
1.2	To account no (TAN)→ Text Box

1.2.1	TAN→ Accept only 4 digit
1.3	Amount→ Text Box
1.3.1	Amount → Accept maximum 4 digit
1.4	Transfer→ Button
1.4.1	Transfer → Enabled
1.5	Cancel→ Button
1.5.1	Cancel→ Enabled

Below are the application access details, which is given by the customer

- URL→ login Page
- Username/password/OK → home page
- To reach Amount transfer module follow the below

Loans → sales → Amount transfer

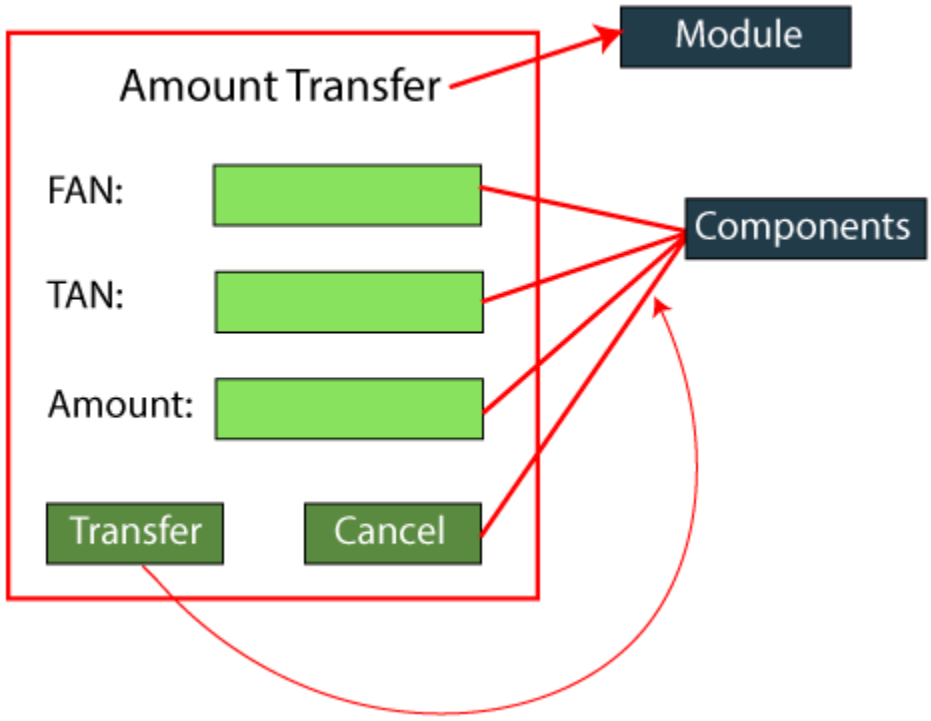
While performing unit testing, we should follow some rules, which are as follows:

- To start unit testing, at least we should have one module.
- Test for positive values
- Test for negative values
- No over testing
- No assumption required

When we feel that the **maximum test coverage** is achieved, we will stop the testing.

Now, we will start performing the unit testing on the different components such as

- **From account number(FAN)**
- **To account number(TAN)**
- **Amount**
- **Transfer**
- **Cancel**



For the FAN components

Values	Description
1234	accept
4311	Error message→ account valid or not
blank	Error message→ enter some values

5 digit/ 3 digit	Error message→ accept only 4 digit
Alphanumeric	Error message → accept only digit
Blocked account no	Error message
Copy and paste the value	Error message→ type the value
Same as FAN and TAN	Error message

For the TAN component

- Provide the values just like we did in **From account number** (FAN) components

For Amount component

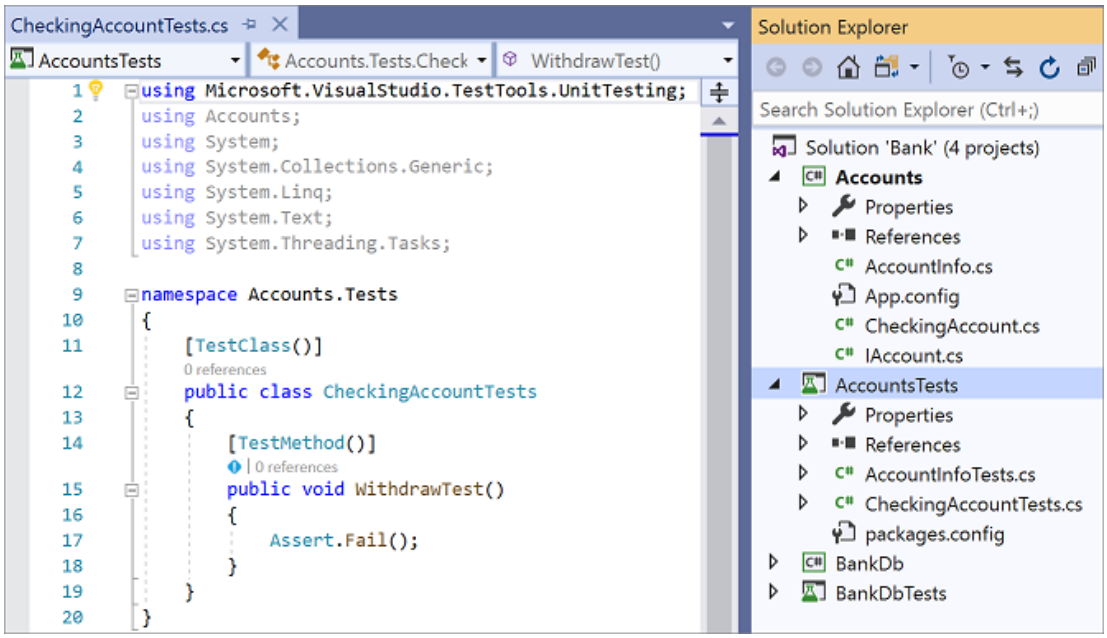
- Provide the values just like we did in FAN and TAN components.

For Transfer component

- Enter valid FAN value
- Enter valid TAN value
- Enter the correct value of Amount
- Click on the Transfer button→ amount transfer successfully(confirmation message)

For Cancel Component

- Enter the values of FAN, TAN, and amount.
- Click on the Cancel button → all data should be cleared.


CheckingAccountTests.cs
AccountsTests
Accounts.Tests.Check
WithdrawTest()
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Accounts;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Accounts.Tests
{
[TestClass()]
0 references
public class CheckingAccountTests
{
[TestMethod()]
0 references
public void WithdrawTest()
{
Assert.Fail();
}
}
}
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'Bank' (4 projects)
Accounts
Properties
References
AccountInfo.cs
App.config
CheckingAccount.cs
IAccount.cs
AccountsTests
Properties
References
AccountInfoTests.cs
CheckingAccountTests.cs
packages.config
BankDb
BankDbTests

LEARNING MATERIAL TO STUDENTS (UNIT TESTING)

What is Unit Testing?

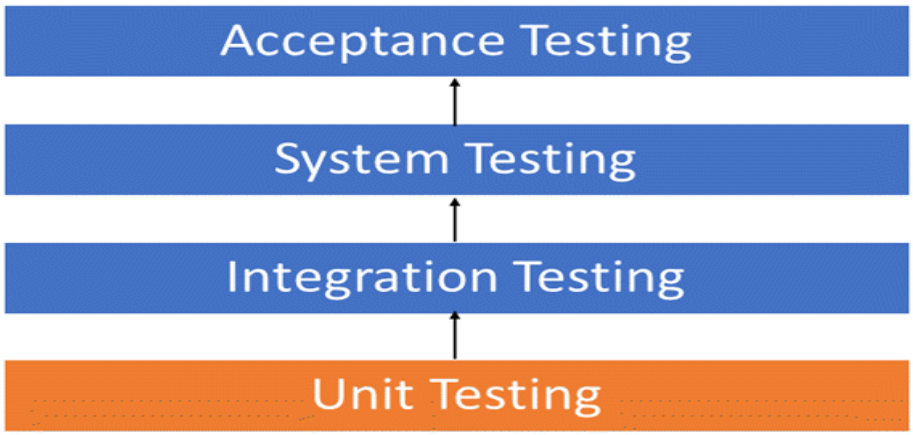
UNIT TESTING is a type of software testing where individual units or components of a software are tested. The purpose is to validate that each unit of the software code performs as expected. Unit Testing is done during the development (coding phase) of an application by the developers. Unit Tests isolate a section of code and verify its correctness. A unit may be an individual function, method, procedure, module, or object.
In SDLC, STLC, V Model, Unit testing is first level of testing done before integration testing. Unit testing is a WhiteBox testing technique that is usually performed by the developer. Though, in a practical world due to time crunch or reluctance of developers to tests, QA engineers also do unit testing.

In this tutorial, you will learn-

Why Unit Testing?

Unit Testing is important because software developers sometimes try saving time doing minimal unit testing and this is myth because inappropriate unit testing leads to high cost Defect fixing during System Testing, Integration Testing and even Beta Testing after application is built. If proper unit testing is done in early development, then it saves time and money in the end.

Here, are the key reasons to perform unit testing in software engineering



Unit Testing Levels

1. Unit tests help to fix bugs early in the development cycle and save costs.
2. It helps the developers to understand the testing code base and enables them to make changes quickly
3. Good unit tests serve as project documentation
4. Unit tests help with code re-use. Migrate both your code **and** your tests to your new project. Tweak the code until the tests run again.

How to do Unit Testing

In order **to do Unit Testing**, developers write a section of code to test a specific function in software application. Developers can also isolate this function to test more rigorously which reveals unnecessary dependencies between function being tested and other units so the dependencies can be eliminated. Developers generally use UnitTest framework to develop automated test cases for unit testing.
Unit Testing is of two types

- Manual
- Automated

Unit testing is commonly automated but may still be performed manually. Software Engineering does not favor one over the other but automation is preferred. A manual approach to unit testing may employ a step-by-step instructional document.

Under the automated approach-

- A developer writes a section of code in the application just to test the function. They would later comment out and finally remove the test code when the application is deployed.
- A developer could also isolate the function to test it more rigorously. This is a more thorough unit testing practice that involves copy and paste of code to its own testing environment than its natural environment. **Isolating the code helps in revealing unnecessary dependencies between the code being tested and other units or data spaces** in the product. These dependencies can then be eliminated.
- A coder generally uses a UnitTest Framework to develop automated test cases. Using an automation framework, the developer codes criteria into the test to verify the correctness of the code. During execution of the test cases, the framework logs failing test cases. Many frameworks will also automatically flag and report, in summary, these failed test cases. Depending on the severity of a failure, the framework may halt subsequent testing.
- The workflow of Unit Testing is 1) Create Test Cases 2) Review/Rework 3) Baseline 4) Execute Test Cases.

Unit Testing Techniques

The **Unit Testing Techniques** are mainly categorized into three parts which are Black box testing that involves testing of user interface along with input and output, White box testing that involves testing the functional behaviour of the software application and Gray box testing that is used to execute test suites, test methods, test cases and performing risk analysis.
Code coverage techniques used in Unit Testing are listed below:

- Statement Coverage
- Decision Coverage
- Branch Coverage
- Condition Coverage
- Finite State Machine Coverage

For more in refer https://www.guru99.com/code-coverage.html

Unit Test Example: Mock Objects

Unit testing relies on mock objects being created to test sections of code that are not yet part of a complete application. Mock objects fill in for the missing parts of the program.

For example, you might have a function that needs variables or objects that are not created yet. In unit testing, those will be accounted for in the form of mock objects created solely for the purpose of the unit testing done on that section of code.

Unit Testing Tools

There are several automated unit test software available to assist with unit testing. We will provide a few examples below:

1. Junit: Junit is a free to use testing tool used for Java programming language. It provides assertions to identify test method. This tool test data first and then inserted in the piece of code.
2. NUnit: NUnit is widely used unit-testing framework use for all .net languages. It is an open source tool which allows writing scripts manually. It supports data-driven tests which can run in parallel.
3. JMockit: JMockit is open source Unit testing tool. It is a code coverage tool with line and path metrics. It allows mocking API with recording and verification syntax. This tool offers Line coverage, Path Coverage, and Data Coverage.
4. EMMA: EMMA is an open-source toolkit for analyzing and reporting code written in Java language. Emma support coverage types like method, line, basic block. It is Java-based so it is without external library dependencies and can access the source code.
5. PHPUnit: PHPUnit is a unit testing tool for PHP programmer. It takes small portions of code which is called units and test each of them separately. The tool also allows developers to use pre-define assertion methods to assert that a system behave in a certain manner.

Those are just a few of the available unit testing tools. There are lots more, especially for C languages and Java, but you are sure to find a unit testing tool for your programming needs regardless of the language you use.

Test Driven Development (TDD) & Unit Testing

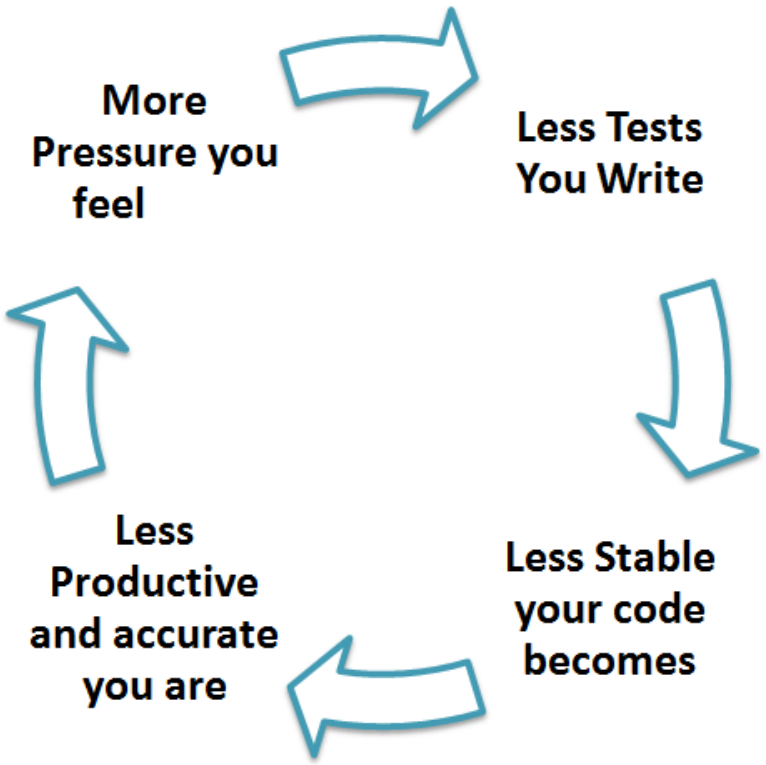
Unit testing in TDD involves an extensive use of testing frameworks. A unit test framework is used in order to create automated unit tests. Unit testing frameworks are not unique to TDD, but they are essential to it. Below we look at some of what TDD brings to the world of unit testing:

- Tests are written before the code
- Rely heavily on testing frameworks
- All classes in the applications are tested
- Quick and easy integration is made possible

Unit Testing Myth

Myth: It requires time, and I am always overscheduled
My code is rock solid! I do not need unit tests.

Myths by their very nature are false assumptions. These assumptions lead to a vicious cycle as follows –



Truth is Unit testing increase the speed of development.

Programmers think that Integration Testing will catch all errors and do not execute the unit test. Once units are integrated, very simple errors which could have very easily found and fixed in unit tested take a very long time to be traced and fixed.

Unit Testing Advantage

- Developers looking to learn what functionality is provided by a unit and how to use it can look at the unit tests to gain a basic understanding of the unit API.

- Unit testing allows the programmer to refactor code at a later date, and make sure the module still works correctly (i.e. Regression testing). The procedure is to write test cases for all functions and methods so that whenever a change causes a fault, it can be quickly identified and fixed.
- Due to the modular nature of the unit testing, we can test parts of the project without waiting for others to be completed.

Unit Testing Disadvantages

- Unit testing can’t be expected to catch every error in a program. It is not possible to evaluate all execution paths even in the most trivial programs
- Unit testing by its very nature focuses on a unit of code. Hence it can’t catch integration errors or broad system level errors.

C. **The work must be reproducible and developed further by other scholars (2)**

This flipped learning demonstration was helpful for students to gain insight into the subject and it was further carried out by **Ms.Nazia Amreen teach testing methods.**

D. Statement of clear goals, use of appropriate methods, significance of results, effective presentation and reflective critique (10)

Goals:

A Goal of teaching with flipped learning is that the students are actively engaged in figuring out the principles by abstracting from the examples. This develops their skills in:

- Problem solving
- Analytical tools, quantitative and/or qualitative, depending on the case
- Decision making in complex situations
- Coping with ambiguities

Appropriate Methods: ICT, PPTs Presentation (Seminars), Online c compiler, Turboo C++.

Significance of Result:

The main objective in mind when teaching this course is to make testing more interesting to students who do not view it so, assuming that greater student interest would lead to better performance in class and deeper understanding and appreciation of the subject. Introductory testing courses often teach such basic concepts that it is difficult to design assignments that are simultaneously simple, challenging and interesting. We feel that our teaching methodology provides all three aspects. Our interactions in the classroom, along with the average performance of the students in the assignments, provide encouraging evidence that our approach worked well in these aspects.

Students were able to get good results both in practical and theory

	Practical Exam	Theory semester end Exam
Pass percentage	100%	83%

Effective presentation and Reflective critique:

The objective of the course is providing a foundation of testing methodologies that student can and will apply to discipline study.

Our goal is to involve student actively in using testing in their discipline and seek a unified approach to teach various non-majors.

To evaluate the effectiveness of our approach, during the course we interacted with student personally and took the response about various aspects regarding the course.

Feedback and Result Analysis

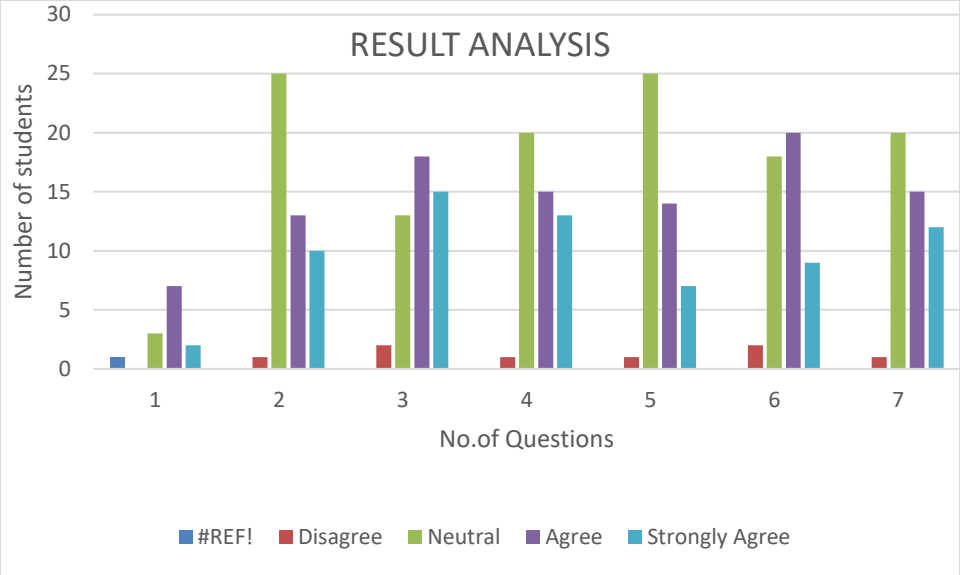
A surveyhas been conducted in which students and faculties were asked certain questions .

Questionnaire/Feedback From Students

S.No	Question No	Questions
1.	**Q1**	The flipped learning increased your knowledge and skills in the subject matter.
2.	**Q2**	The course gave you the confidence to do more advanced work in the subject
3.	**Q3**	Do you believe that what you are being asked to learn in this course is important
4.	**Q4**	Overall, this course met your expectations for the quality of the course
5.	**Q5**	The course was helpful in progress toward my degree
6.	**Q6**	**The instructor's teaching methods were effective.**
7.	**Q7**	**The instructor encouraged student participation in class.**

Distribution of students' answers (numbers and percentage) provided for the survey's questions

Question No	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
Q1	1	2	20	15	12
Q2	1	1	25	13	10
Q3	2	2	13	18	15
Q4	1	1	20	15	13
Q5	2	1	25	14	7
Q6	1	2	18	20	9
Q7	1	1	20	15	12



Questionnaire/Feedback from Faculty/Peer Review

S.No	Question No	Questions
1.	**Q1**	The instructor effectively explained and illustrated the purpose and importance of the course concepts
2.	**Q2**	The instructor communicated clearly and was easy to understand i.e., teaching methods were effective.
3.	**Q3**	The instructor effectively organized and facilitated well-run learning activities and was well-prepared for class/discussion sections.
4.	**Q4**	Did this Innovative practice helped to improve and increase the abilities of students.
5.	**Q5**	The instructor cared about the students, their progress, and successful course completion.
6.	**Q6**	I would highly recommend this instructor to other students as it

		encouraged student participation in class.

Distribution of faculties' answers (numbers and percentage) provided for the survey's questions

Question No	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
Q1	0	0	5	2	5
Q2	0	0	2	6	4
Q3	0	0	3	6	3
Q4	0	0	2	5	5
Q5	0	0	1	5	6
Q6	0	0	1	3	8