UNIT - 3

❖ software construction

The term software construction refers to the detailed creation of working software through a combination of coding, verification, unit testing, integration testing, and debugging. The Software Construction knowledge area (KA) is linked to all the other KAs, but it is most strongly linked to Software Design and Software Testing because the software construction process involves significant software design and testing. The process uses the design output and provides an input to testing ("design" and "testing" in this case referring to the activities, not the KAs). Boundaries between design, construction, and testing (if any) will vary depending on the software life cycle processes that are used in a project. Although some detailed design may be performed prior to construction, much design work is performed during the construction activity.

It is also related to project management, insofar as the management of construction can present considerable challenges.

The importance of software construction

1) Construction is a large part of software development. Depending on the size of the project, construction typically takes 30 to 80 percent of total time spent on a project.
2) Construction is the central activity in software development
    - Requirement and architecture
    - Construction
    - System Testing

SOFTWARE CONSTRUCTION FUNDAMENTALS

Software Construction fundamentals includes:

- minimizing complexity
- anticipating change
- constructing for verification
- reuse
- standards in construction.

The first four concepts apply to design as well as to construction. The following sections define these concepts and describe how they apply to construction.

Minimizing Complexity

Most people are limited in their ability to hold complex structures and information in their working memories, especially over long periods of time. This proves to be a major factor influencing how people convey intent to computers and leads to one of the strongest drives in software construction: minimizing complexity. The need to reduce complexity applies to essentially every aspect of software construction and is particularly critical to testing of software constructions. In software construction, reduced complexity is achieved through emphasizing code creation that is simple and readable rather than clever. It is accomplished through making use of standards, modular design, and numerous other specific techniques. It is also supported by construction-focused quality techniques.

Anticipating Change

Most software will change over time, and the anticipation of change drives many aspects of software construction; changes in the environments in which software operates also affect software in diverse ways. Anticipating change helps software engineers build extensible software, which means they can enhance a software product without disrupting the underlying structure. Anticipating change is supported by many specific techniques .

Constructing for Verification

Constructing for verification means building software in such a way that faults can be readily found by the software engineers writing the software as well as by the testers and users during independent testing and operational activities. Specific techniques that support constructing for verification include following coding standards to support code reviews and unit testing, organizing code to support automated testing, and restricting the use of complex or hard-to- understand language structures, among others.

Reuse

Reuse refers to using existing assets in solving different problems. In software construction, typical assets that are reused include libraries, modules, components, source code, and commercial off-the-shelf (COTS) assets. Reuse is best practiced systematically, according to a well-defined, repeatable process. Systematic reuse can enable significant software productivity, quality, and cost improvements. Reuse has two closely related facets:"construction for reuse" and "construction with reuse." The former means to create reusable software assets, while the latter means to reuse

software assets in the construction of a new solution. Reuse often transcends the boundary of projects, which means reused assets can be constructed in other projects or organizations.

Standards in Construction

Applying external or internal development standards during construction helps achieve a project's objectives for efficiency, quality, and cost. Specifically, the choices of allowable programming language subsets and usage standards are important aids in achieving higher security. Standards that directly affect construction issues include

- communication methods (for example, standards for document formats and contents)
- programming languages (for example, language standards for languages like Java and C++)
- coding standards (for example, standards for naming conventions, layout, and indentation)
- platforms (for example, interface standards for operating system calls)
- tools (for example, diagrammatic standards for notations like UML (Unified Modeling Language)).

❖ Object Oriented Principles in OOAD

Object-oriented principles are a set of guidelines for designing and implementing software systems that are based on the idea of objects. Objects are self-contained units of code that have both data and behavior. They can interact with each other to perform tasks.

Object-Oriented Analysis and Design (OOAD) is a software engineering methodology that uses object-oriented principles to design and implement software systems. OOAD involves a number of techniques and practices, including:

- **Object-Oriented Modelling:** This involves using visual diagrams to represent the different objects in a software system and their relationships to each other.
- **Use Cases:** This involves describing the different ways in which users will interact with a software system.

- **Design Patterns:** This involves using reusable solutions to common problems in software design.

Important Topics for Object Oriented Principles in OOAD

- Abstraction
- Encapsulation
- Modularity
- Hierarchy
- Typing
- Concurrency

1. Abstraction

Think of a TV remote control. It has buttons like power, volume up, volume down, and channel change. Now, let's use this as an example of Abstraction

In OOP, abstraction is like using a TV remote without knowing how it works on the inside. You don't need to know about the wires, circuits, or tiny components inside the remote. All you care about are the buttons and what they do.

In this example:

- **Buttons** are like the functions or actions in a program, such as play, pause, or stop.
- **What the buttons do** is like the behavior of objects or classes in OOP. For example, when you press the volume up button, the volume goes up,, but you don't need to understand how it happens.

So, **Abstraction** in OOP is about using objects or classes (like our TV remote) without worrying about how they work internally. You only care about what they can do and how to use them, just like using a TV remote without needing to be an electrical engineer to make it work.

Advantages of Abstraction

- Abstraction makes things simpler. It helps us focus on what's important and ignore what's not, making it easier to understand.
- We can reuse the same template for different things, saving time and making our work more efficient.

- When something goes wrong or needs an update, we can fix just the part that's broken without messing up everything else.
- Abstraction helps us grow our projects without making them messy.

Disadvantages of Abstraction

- Sometimes, too much abstraction can make things more hard to understand.
- Abstraction can add extra work and code that might slow down our program a little bit.
- Using abstraction tools can be tricky for beginners.

2. Encapsulation

Let us take an example of a water bottle to explain encapsulation:

- **The Bottle:** In OOP, a class is like the bottle, with visible features (attributes and methods) and hidden contents (data and functions).
- **The Cap:** The cap is like encapsulation. It protects what's inside the bottle (the object) and keeps it safe from outside interference.
- **The Water:** Inside the bottle is data, like water. You can use the bottle (object) to access and modify the data, without needing to know how it's stored or processed inside.

So, encapsulation in OOP is like a cap on water bottle, keeping the inner workings hidden and secure. letting you use the object without worrying about its internal details.

Advantages of Encapsulation

- Encapsulation keeps data safe and prevents unauthorized access or modification.
- It allows controlled access to data through methods.
- You can change the internal workings of an object without affecting the code that uses it.
- Encapsulation makes code easier to understand and maintain because you only need to focus on what a class does, not how it does it.

Disadvantages of Encapsulation

- It can add extra layers to your code, making it a bit more complex.

- Encapsulation can sometimes make your code a bit slower because of the extra control it imposes.

3. Modularity

Modularity in OOAD is like organizing your kitchen. Just as you keep pots in one cabinet and dishes in another for easier access and maintenance, in OOAD, you group similar functions and data into organized module or classes. This makes it simpler to understand and change specific parts of your software without affecting the entire program, similar to how you can upgrade one appliance in your kitchen without redoing the whole room.

Advantages of Modularity

- Modularity makes it easier to fix or update one part of a software system with messing up the rest.
- You can reuse modules in different parts of your software, saving time and effort.
- Multiple programmers can work on different modules simultaneously.

Disadvantages of Modularity

- Overdoing modularity can make your software too complex with many small parts, making it harder to grasp.
- Breaking a program into modules may add some extra work and slow down the software slightly.

4. Hierarchy

Lets us take an example of a family. In a family there are grandparents, parents, and children:

- **Classes as Family Members:** In OOAD, think of your family members as classes or objects. Each class has a specific job, like each family members has a role in the family.
- **Family Hierarchy:** Just like your family tree has a hierarchy with grandparents at the top, parents in the middle and children at the bottom, in OOAD classes can can be organized in a hierarchy. Some classes are more general (like parents) and others are more specific (like children).
- **Inheritance:** Imagine your grandparents passing down family traditions to your parents, who then pass then on to you. This is similar in OOAD where classes can inherit features from other classes higher up in the hierarchy.

- **Specialization:** You and your siblings have more specialized roles compared to your parents. This is like specialization in OOAD, where subclasses have specific features compared to their parent classes.

So, hierarchy in OOAD is like arranging classes in an organized way, just as your family tree helps you understand your family's structure. It helps in managing and understanding the relationships between different classes in a software system.

5. Typing

Typing involves categorizing objects based on their data types (e.g., integers, strings, custon objects) to ensure they are used appropriately.

Example:

Think about sorting your belongings. You wouldn't mix up your books, clothes, and kitchen utensils in the same box. Similarly, in programming, you categorize data based on their data types (e.g., numbers, text, dates) to perform operations correctly. This helps to prevent errors and make code more readable and maintainable.

6. Concurrency

Concurrency in Object Oriented Analysis and Design (OOAD) is like managing multiple tasks at at the same time, just as people multitask in every day life.

Imagine you're a chef in a restaurant. You have several orders to prepare, and each order consists of different dishes. You can't cook one disk at a time and move to the next dish because customers are hungry and waiting for their food. So, you need to work on multiple dishes simultaneously

Now, let's relate this to OOAD:

- **Tasks as Objects:** In OOAD, think of each dish you're cooking as an object or a task. Each dish has its recipe ad cooking instructions, just like objects have their method and properties.

- **Concurrency in Kitchen:** You are working concurrently in kitchen, managing multiple dishes simultaneously. While one dish is simmering, you might be chopping ingredients for another or seasoning a third. You switch between tasks efficiently to serve all orders.

In OOAD, concurrency is about managing multiple tasks or processes within a software system simultaneously. It's like juggling different tasks efficiently to make the most of your time.

❖ Object-oriented programming Languages

Object-oriented programming (**OOP**) is a programming paradigm based on the concept of *objects*, which can contain data and code: data in the form of fields (often known as attributes or properties), and code in the form of procedures (often known as methods). In OOP, computer programs are designed by making them out of objects that interact with one another.

Many of the most widely used programming languages (such as C++, Java, and Python) are multi-paradigm and support object-oriented programming to a greater or lesser degree, typically in combination with imperative programming and declarative programming

1. C++ Programming

C++ is an object-oriented programming language. It is an extension to C programming.

Our C++ tutorial includes all topics of C++ such as first example, control statements, objects and classes, inheritance, constructor, destructor, this, static, polymorphism, abstraction, abstract class, interface, namespace, encapsulation, arrays, strings, exception handling, File IO, etc.

C++ programming language was developed in 1980 by Bjarne Stroustrup at bell laboratories of AT&T (American Telephone & Telegraph), located in U.S.A.

C++ is a general purpose, case-sensitive, free-form programming language that supports object-oriented, procedural and generic programming.

C++ is a middle-level language, as it encapsulates both high and low level language features.

C++ supports the object-oriented programming, the four major pillar of object-oriented programming (OOPs) used in C++ are:

1. Inheritance
2. Polymorphism
3. Encapsulation
4. Abstraction

Usage of C++

By the help of C++ programming language, we can develop different types of secured and robust applications:

- Window application
- Client-Server application
- Device drivers
- Embedded firmware etc

C++ Program

File: main.cpp

```
#include <iostream>
using namespace std;
int main() {
  cout << "Hello C++ Programming";
  return 0;
}
```

2. Java Programming

Java is an object-oriented, class-based, concurrent, secured and general-purpose computer-programming language. It is a widely used robust technology.

Java is a **programming language** and a **platform**. Java is a high level, robust, object-oriented and secure programming language.

Java was developed by *Sun Microsystems* (which is now the subsidiary of Oracle) in the year 1995. *James Gosling* is known as the father of Java. Before Java, its name was *Oak*. Since Oak was already a registered company, so James Gosling and his team changed the name from Oak to Java.

Java Example

Let's have a quick look at Java programming example. A detailed description of Hello Java example is available in next page.

Simple.java

```
class Simple{
    public static void main(String args[]){
     System.out.println("Hello Java");
    }
}
```

Application

According to Sun Microsystems, 3 billion devices run Java. There are various devices where Java is currently used. Some of them are as follows:

1. Desktop Applications such as acrobat reader, media player, antivirus, etc.
2. Web Applications such as irctc.co.in, javatpoint.com, etc.
3. Enterprise Applications such as banking applications.
4. Mobile
5. Embedded System
6. Smart Card
7. Robotics
8. Games, etc.

3. Python Programming

Python is a widely used programming language that offers several unique features and advantages compared to languages like **Java** and **C++.** Our Python tutorial thoroughly explains Python basics and advanced concepts, starting with installation, conditional statements, loops, built-in data structures, Object-Oriented Programming, Generators, Exception Handling, Python RegEx, and many other concepts. This tutorial is designed for beginners and working professionals.

In the late 1980s, Guido van Rossum dreamed of developing Python. The first version of **Python 0.9.0 was released in 1991**. Since its release, Python started gaining popularity. According to reports, Python is now the most popular programming language among developers because of its high demands in the tech realm.

In February 1991, the first public version of Python, version 0.9.0, was released. This marked the official birth of **Python as an open-source project**. The language was named after the British comedy series "**Monty Python's Flying Circus**".

Python is a general-purpose, dynamically typed, high-level, compiled and interpreted, garbage-collected, and purely object-oriented programming language that supports procedural, object-oriented, and functional programming.

Python Code:

```
print("Hello World!")
```

Python Applications

As per a survey it is observed that ***Python is the main coding language for more than 80% of developers***. The main reason behind this is its extensive libraries and frameworks that fuel up the process

1. Web Development
2. Machine Learning and Artificial Intelligence
3. Data Science
4. Game Development
5. Audio and Visual Applications
6. Software Development
7. CAD Applications
8. Business Applications
9. Desktop GUI
10. Web Scraping Application

❖ SOFTWARE TESTING

Software Testing is evaluation of the software against requirements gathered from users and system specifications. Testing is conducted at the phase level in software development life cycle or at module level in program code. Software testing comprises of Validation and Verification.

Validation:

Validation is process of examining whether or not the software satisfies the user requirements. It is carried out at the end of the SDLC. If the software matches requirements for which it was made, it is validated.

- Validation ensures the product under development is as per the user requirements.

- Validation answers the question – "Are we developing the product which attempts all that user needs from this software ?".
- Validation emphasizes on user requirements.

Verification:

Verification is the process of confirming if the software is meeting the business requirements, and is developed adhering to the proper specifications and methodologies.

- Verification ensures the product being developed is according to design specifications.
- Verification answers the question– "Are we developing this product by firmly following all design specifications ?"
- Verifications concentrates on the design and system specifications.

Classification of Software Testing:

Software Testing can be broadly classified into two types:

- ✓ **Manual:**

This testing is performed without taking help of automated testing tools. The software tester prepares test cases for different sections and levels of the code, executes the tests and reports the result to the manager.

Manual testing is time and resource consuming. The tester needs to confirm whether or not right test cases are used. Major portion of testing involves manual testing.

- ✓ **Automated:**

This testing is a testing procedure done with aid of automated testing tools. The limitations with manual testing can be overcome using automated test tools.

Testing approaches:

Tests can be conducted based on two approaches –

1. **Functionality testing/Black-box testing:**

- The technique of testing in which the tester doesn't have access to the source code of the software and is conducted at the software interface without any concern with the internal logical structure of the software is known as blackbox testing.

- It is carried out to test functionality of the program. It is also called 'Behavioral' testing.
- In this testing method, the design and structure of the code are not known to the tester, and testing engineers and end users conduct this test on the software.

Black-box testing techniques:

i. **Equivalence class :**

The input is divided into similar classes. If one element of a class passes the test, it is assumed that all the class is passed.

ii. **Boundary values :**

The input is divided into higher and lower end values. If these values pass the test, it is assumed that all values in between may pass too.

iii. **Cause-effect graphing:**

In both previous methods, only one input value at a time is tested. Cause (input) – Effect (output) is a testing technique where combinations of input values are tested in a systematic way.

iv. **Pair-wise Testing :**

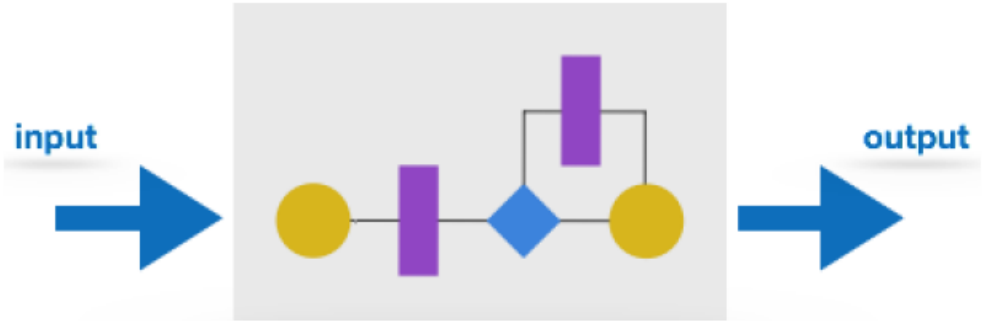
The behavior of software depends on multiple parameters. In pairwise testing, the multiple parameters are tested pair-wise for their different values.

v. **State-based testing :**

The system changes state on provision of input. These systems are tested based on their states and input.

2. **Implementation testing/White-box testing:**

- The technique of testing in which the tester is aware of the internal workings of the product, has access to its source code, and is conducted by making sure that all internal operations are performed according to the specifications is known as white box testing.



- It is conducted to test program and its implementation, in order to improve code efficiency or structure. It is also known as 'Structural' testing.
- In this testing method, the design and structure of the code are known to the tester. Programmers of the code conduct this test on the code.

White-box testing techniques:

I. **Control-flow testing -** The purpose of the control-flow testing to set up test cases which covers all statements and branch conditions. The branch conditions are tested for both being true and false, so that all statements can be covered.
II. **Data-flow testing -** This testing technique emphasis to cover all the data variables included in the program. It tests where the variables were declared and defined and where they were used or changed.

❖ Unit Testing

Unit Testing is a software testing technique in which individual units or components of a software application are tested in isolation. These units are the smallest pieces of code, typically functions or methods, ensuring they perform as expected.

Unit testing helps in identifying bugs early in the development cycle, enhancing code quality, and reducing the cost of fixing issues later. It is an essential part of **Test-Driven Development (TDD)**, promoting reliable code.

Unit testing is the process of testing the smallest parts of your code, like individual functions or methods, to make sure they work correctly. It's a key part of software development that improves code quality by testing each unit in isolation.

Unit testing strategies

To create effective **unit tests**, follow these basic techniques to ensure all scenarios are covered:

- **Logic checks**: Verify if the system performs correct calculations and follows the expected path with valid inputs. Check all possible paths through the code are tested.
- **Boundary checks**: Test how the system handles typical, edge case, and invalid inputs. For example, if an integer between 3 and 7 is expected, check how the system reacts to a 5 (normal), a 3 (edge case), and a 9 (invalid input).
- **Error handling**: Check the system properly handles errors. Does it prompt for a new input, or does it crash when something goes wrong?
- **Object-oriented checks**: If the code modifies objects, confirm that the object's state is correctly updated after running the code.

Benefits of unit testing

Here are the Unit testing benefits which used in the software development with many ways:

1. **Early Detection of Issues:** Unit testing allows developers to detect and fix issues early in the development process before they become larger and more difficult to fix.
2. **Improved Code Quality:** Unit testing helps to ensure that each unit of code works as intended and meets the requirements, improving the overall quality of the software.
3. **Increased Confidence:** Unit testing provides developers with confidence in their code, as they can validate that each unit of the software is functioning as expected.
4. **Faster Development:** Unit testing enables developers to work faster and more efficiently, as they can validate changes to the code without having to wait for the full system to be tested.

5. **Better Documentation:** Unit testing provides clear and concise documentation of the code and its behavior, making it easier for other developers to understand and maintain the software.

6. **Facilitation of Refactoring:** Unit testing enables developers to safely make changes to the code, as they can validate that their changes do not break existing functionality.

7. **Reduced Time and Cost:** Unit testing can reduce the time and cost required for later testing, as it helps to identify and fix issues early in the development process.

Disadvantages of Unit Testing

1. **Time and Effort:** Unit testing requires a significant investment of time and effort to create and maintain the test cases, especially for complex systems.

2. **Dependence on Developers:** The success of unit testing depends on the developers, who must write clear, concise, and comprehensive test cases to validate the code.

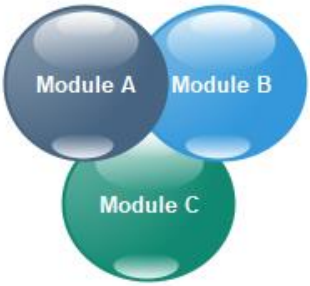
3. **Difficulty in Testing Complex Units:** Unit testing can be challenging when dealing with complex units, as it can be difficult to isolate and test individual units in isolation from the rest of the system.

4. **Difficulty in Testing Interactions:** Unit testing may not be sufficient for testing interactions between units, as it only focuses on individual units.

❖ Integration testing

Integration testing is the process of testing the interface between two software units or modules. It focuses on determining the correctness of the interface. The purpose of integration testing is to expose faults in the interaction between integrated units. Once all the modules have been unit-tested, integration testing is performed.

Tested in Unit Testing



Under Integration Testing

Integration testing is a [software testing technique]that focuses on verifying the interactions and data exchange between different components or modules of a software application. The goal of integration testing is to identify any problems or bugs that arise when different components are combined and interact with each other. Integration testing is typically performed after unit testing and before system testing. It helps to identify and resolve integration issues early in the development cycle, reducing the risk of more severe and costly problems later on.

Approaches of Integration Testing

Incremental integration testing can be further divided into 3 smaller approaches, each also comes with its own advantages and disadvantages that QA teams need to carefully consider for their projects. These approaches are named based on the level of impact of the software components being integrated have on the overall system, including:

- **Bottom-up approach:** perform testing for low-level components first, then gradually move to higher-level components.
- **Top-down approach:** perform testing for high-level components first, then gradually move to lower-level components.
- **Hybrid approach:** combining the two former approaches

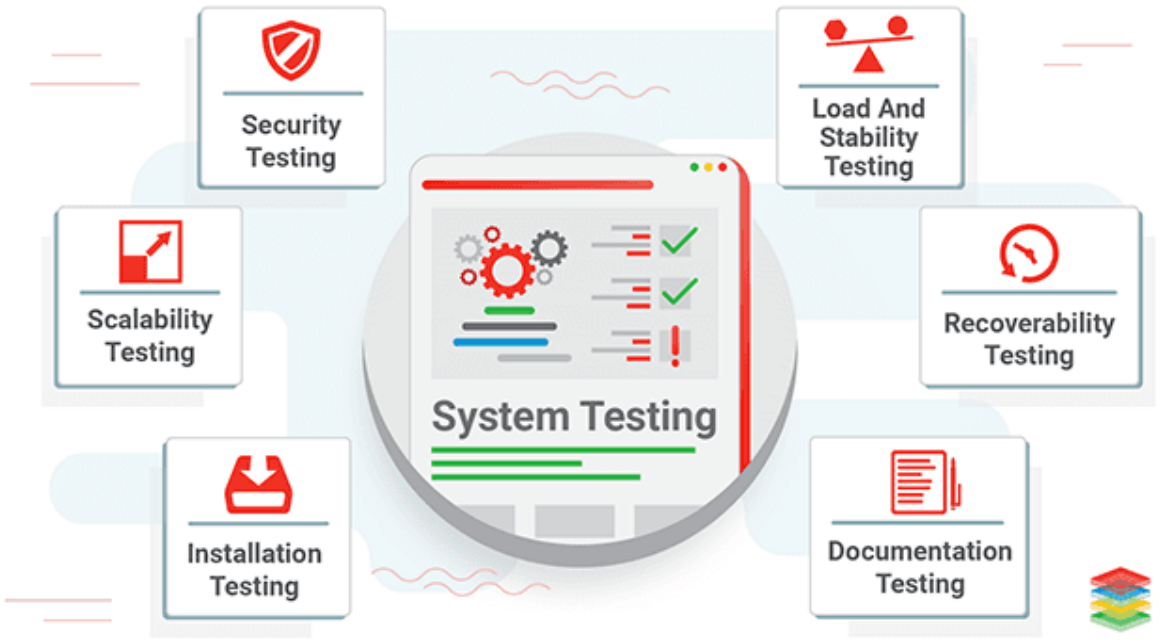
To better understand these 3 concepts, we must first define **low-level components** and **high-level components**.

Applications of Integration Testing

1. **Identify the components:** Identify the individual components of your application that need to be integrated. This could include the frontend, backend, database, and any third-party services.
2. **Create a test plan:** Develop a test plan that outlines the scenarios and test cases that need to be executed to validate the integration points between the different components. This could include testing data flow, communication protocols, and error handling.
3. **Set up test environment:** Set up a test environment that mirrors the production environment as closely as possible. This will help ensure that the results of your integration tests are accurate and reliable.
4. **Execute the tests:** Execute the tests outlined in your test plan, starting with the most critical and complex scenarios. Be sure to log any defects or issues that you encounter during testing.

❖ System testing

System testing is a type of software testing that evaluates the overall functionality and performance of a complete and fully integrated software solution. It tests if the system meets the specified requirements and if it is suitable for delivery to the end-users. This type of testing is performed after the integration testing and before the acceptance testing.



System Testing is a type of software testing that is performed on a completely integrated system to evaluate the compliance of the system with the corresponding requirements. In system testing, integration testing passed components are taken as input.

- The goal of integration testing is to detect any irregularity between the units that are integrated. System testing detects defects within both the integrated units and the whole system. The result of system testing is the observed behavior of a component or a system when it is tested.
- **System Testing** is carried out on the whole system in the context of either system requirement specifications or functional requirement specifications or the context of both. System testing tests the design and behavior of the system and also the expectations of the customer.
- It is performed to test the system beyond the bounds mentioned in the software requirements specification (SRS) . System Testing is performed by a testing team that is independent of the development team and helps to test the quality of the system impartial.
- It has both functional and non-functional testing. **System Testing is a black-box testing** . System Testing is performed after the integration testing and before the acceptance testing.

Types of System Testing

- **Performance Testing:** Performance Testing is a type of software testing that is carried out to test the speed, scalability, stability and reliability of the software product or application.
- **Load Testing:** Load Testing is a type of software Testing which is carried out to determine the behavior of a system or software product under extreme load.
- **Stress Testing:** Stress Testing is a type of software testing performed to check the robustness of the system under the varying loads.
- **Scalability Testing:** Scalability Testing is a type of software testing which is carried out to check the performance of a software application or system in terms of its capability to scale up or scale down the number of user request load.

❖ Test Driven Development (TDD)

Test Driven Development (TDD) is a software development practice where developers write automated tests before writing the actual code that needs to be tested. Developers create unit test cases before developing the actual code. It is an iterative approach combining Programming, Unit Test Creation, and Refactoring.

The process follows a repetitive cycle known as **Red-Green-Refactor.**

1. **Red Phase**: First, a developer writes a test that defines a desired feature or behavior (the "Red" phase, as the test will fail initially).
2. **Green Phase**: Then, they write the minimum code necessary to pass the test (the "Green" phase).
3. **Refactor**: Finally, the code is refactored for optimization while ensuring the test still passes.

TDD helps ensure that the codebase remains reliable and bug-free by catching errors early in the development process. It promotes better design decisions, as writing tests first forces developers to think more clearly about the functionality they are implementing.

Furthermore, because tests are an integral part of the development process, TDD leads to higher code coverage and makes future modifications or refactoring easier and safer, knowing that existing functionality is thoroughly tested.

- The TDD approach originates from the Agile manifesto principles and Extreme programming.
- As the name suggests, the test process drives software development.
- Moreover, it's a structuring practice that enables developers and testers to obtain optimized code that proves resilient in the long term.
- In TDD, developers create small test cases for every feature based on their initial understanding. The primary intention of this technique is to modify or write new code only if the tests fail. This prevents duplication of test scripts.

Test Driven Development (TDD) Examples

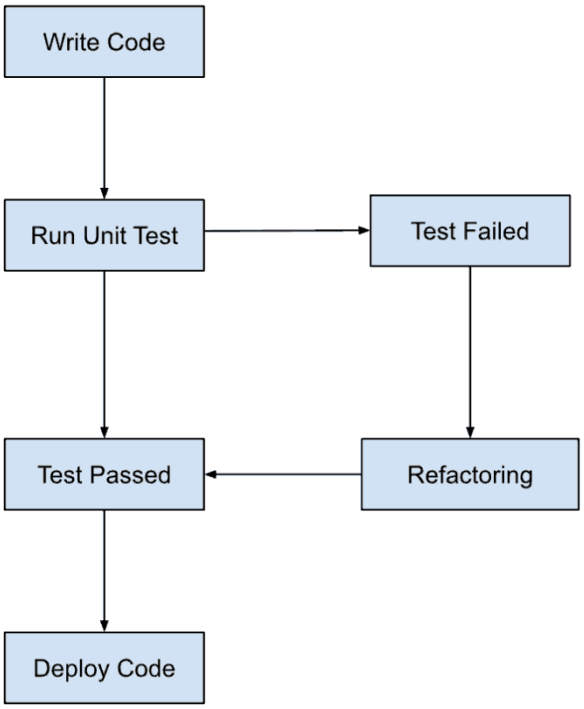
Here are some of the examples where TDD is used:

1. **Calculator Function**: When building a calculator function, a TDD approach would involve writing a test case for the "add" function and then writing the code for the process to pass that test. Once the "add" function is working correctly, additional test cases would be written for other functions such as "subtract", "multiply" and "divide".
2. **User Authentication**: When building a user authentication system, a TDD approach would involve writing a test case for the user login functionality and then writing the code for the login process to pass that test. Once the login functionality works correctly, additional test cases will be written for registration, password reset, and account verification.
3. **E-commerce Website**: When building an e-commerce website, a TDD approach would involve writing test cases for various features such as product listings, shopping cart functionality, and checkout process. Tests would be written to ensure the system works correctly at each process stage, from adding items to the cart to completing the purchase.

Three Phases of Test Driven Development

1. **Create precise tests:** Developers need to create exact unit tests to verify the functionality of specific features. They must ensure that the test compiles so that it can execute. In most cases, the test is bound to fail. This is a meaningful failure as developers create compact tests based on their assumptions of how the feature will behave.
2. **Correcting the Code:** Once a test fails, developers must make the minimal changes required to update the code to run successfully when re-executed.
3. **Refactor the Code:** Once the test runs successfully, check for redundancy or any possible code optimizations to enhance overall performance. Ensure that refactoring does not affect the external behavior of the program.

The image below represents a high-level TDD approach toward development:



Benefits of Test Driven Development (TDD)

1. Fosters the creation of optimized code.
2. It helps developers better analyze and understand client requirements and request clarity when not adequately defined.
3. Adding and testing new functionalities become much easier in the latter stages of development.
4. Test coverage under TDD is much higher compared to conventional development models. The TDD focuses on creating tests for each functionality right from the beginning.
5. It enhances the productivity of the developer and leads to the development of a codebase that is flexible and easy to maintain.