

# **CHƯƠNG 5:**

## **UNIT TESTING**

**Bộ môn Công Nghệ Phần Mềm  
Khoa CNTT – Trường ĐH Ngoại Ngữ Tin Học TP.HCM**

# OBJECTIVES

## HELP STUDENTS:

### UNDERSTANDING:

- WHAT/WHY/WHEN/WHO/HOW TO DO UNIT TESTING
- UNIT TEST FRAMEWORK
- UNIT TEST TECHNIQUES

# WHAT?

- Unit testing is a software development process in which the smallest testable parts of an application , called units, are individually and independently scrutinized for proper operation.
- The main objective of unit testing is to isolate written code to test and determine if it works as intended.
- Unit testing is an important step in the development process, because if done correctly, it can help detect early flaws in code which may be more difficult to find in later testing stages.

# Purpose?

- TO VERIFY THE ACCURACY OF A SECTION OF CODE
- TO HAVE SEPARATE INDEPENDENT SECTIONS OF A CODE
- TO LOCATE AND ADDRESS BUGS EARLY IN SOFTWARE DEVELOPMENT
- TO INCREASE THE PROGRAMMER'S UNDERSTANDING OF THE CODE BASE
- TO BE ABLE TO EFFECT CHANGES EASILY
- TO MAKE CODE REUSABILITY MORE FEASIBLE

- Unit testing is a component of test-driven development (TDD), a pragmatic methodology that takes a meticulous approach to building a product by means of continual testing and revision. This testing method is also the first level of software testing, which is performed before other testing methods such as integration testing. Unit tests are typically isolated to ensure a unit does not rely on any external code or functions. Testing can be done manually but is often automated.
- In SDLC, STLC, V model, unit testing is first level of testing done before integration testing.

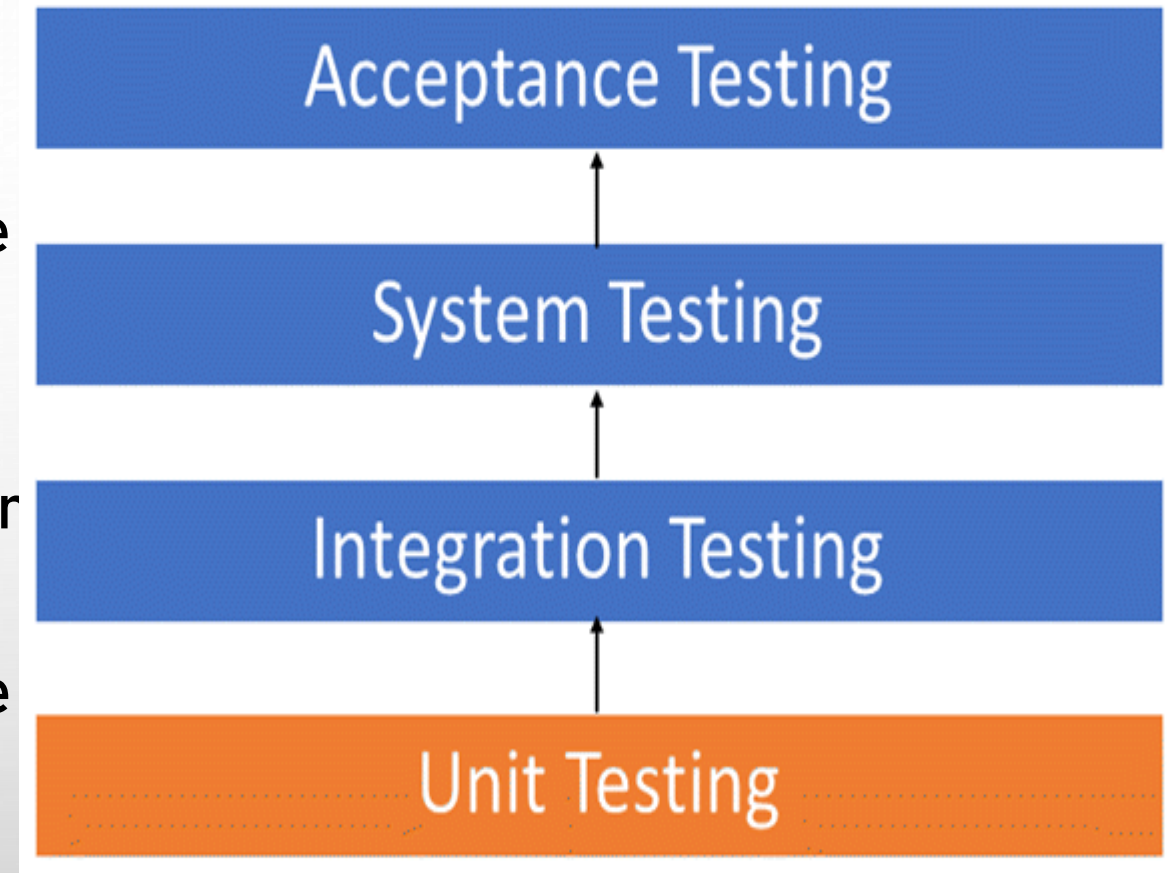
# WHEN & WHO?

- In early stage of SDLC
  - Level 1 of testing
  - In coding stage
  - Developer/ team members who make the code and then test it after or at the time making coding



# WHY?

- 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:






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.



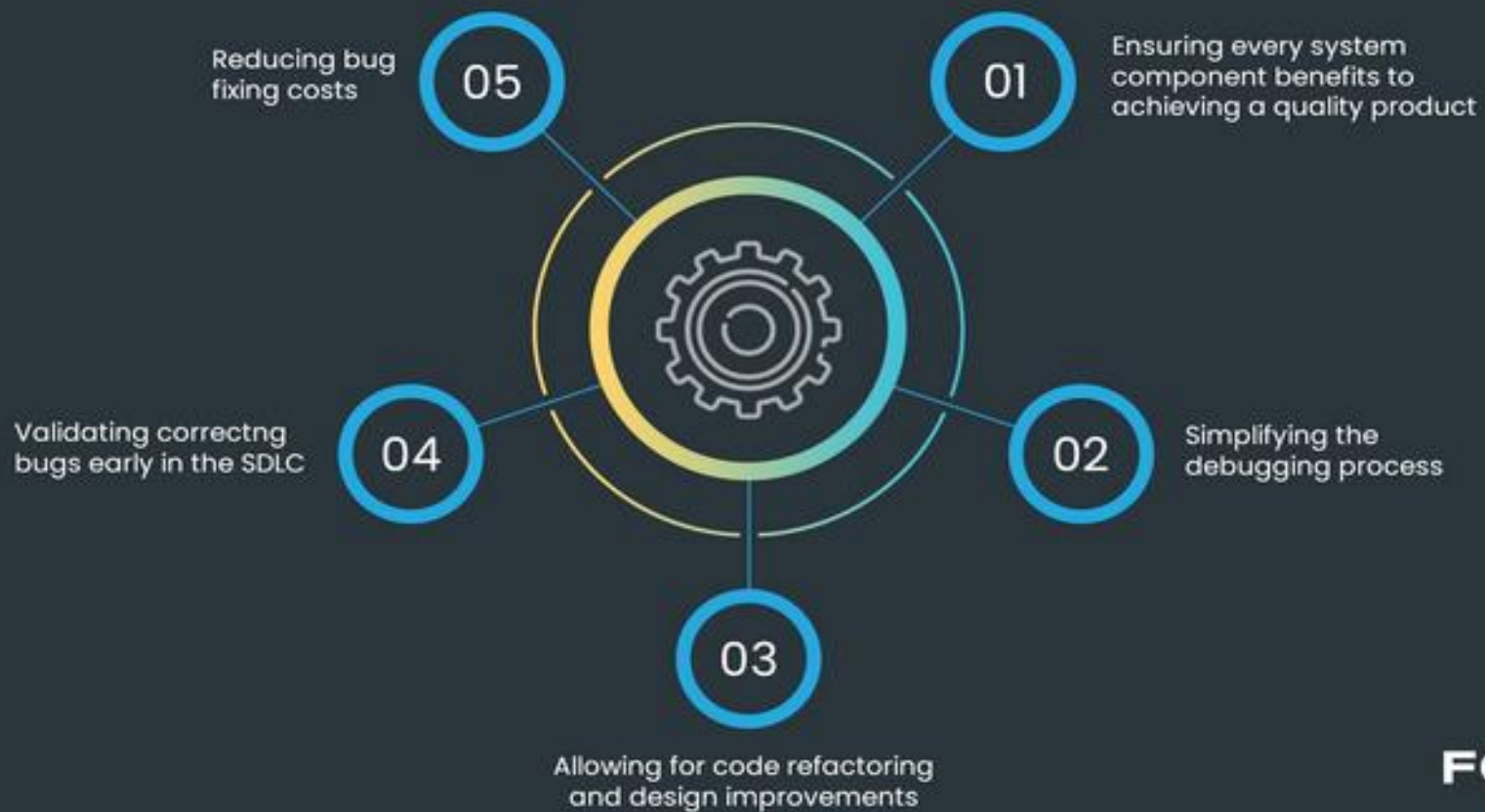


# **BENEFIT?**

- **ANY BUGS ARE FOUND EASILY AND QUICKER**
- **UNIT TESTING SAVES TIME AND MONEY**
- **UNIT TESTING PROVIDES DOCUMENTATION**
- **R2: REUSABLE AND RELIABLE**
- **UNIT TESTING HELPS GAUGE PERFORMANCE**
- **UNIT TESTING IMPROVES CODE COVERAGE**
- **UNIT TESTING REDUCES CODE COMPLEXITY**
- **THE PROCESS BECOMES AGILE**
- **UNIT TESTING RESULTS IN QUALITY SOFTWARE**

# WHY?

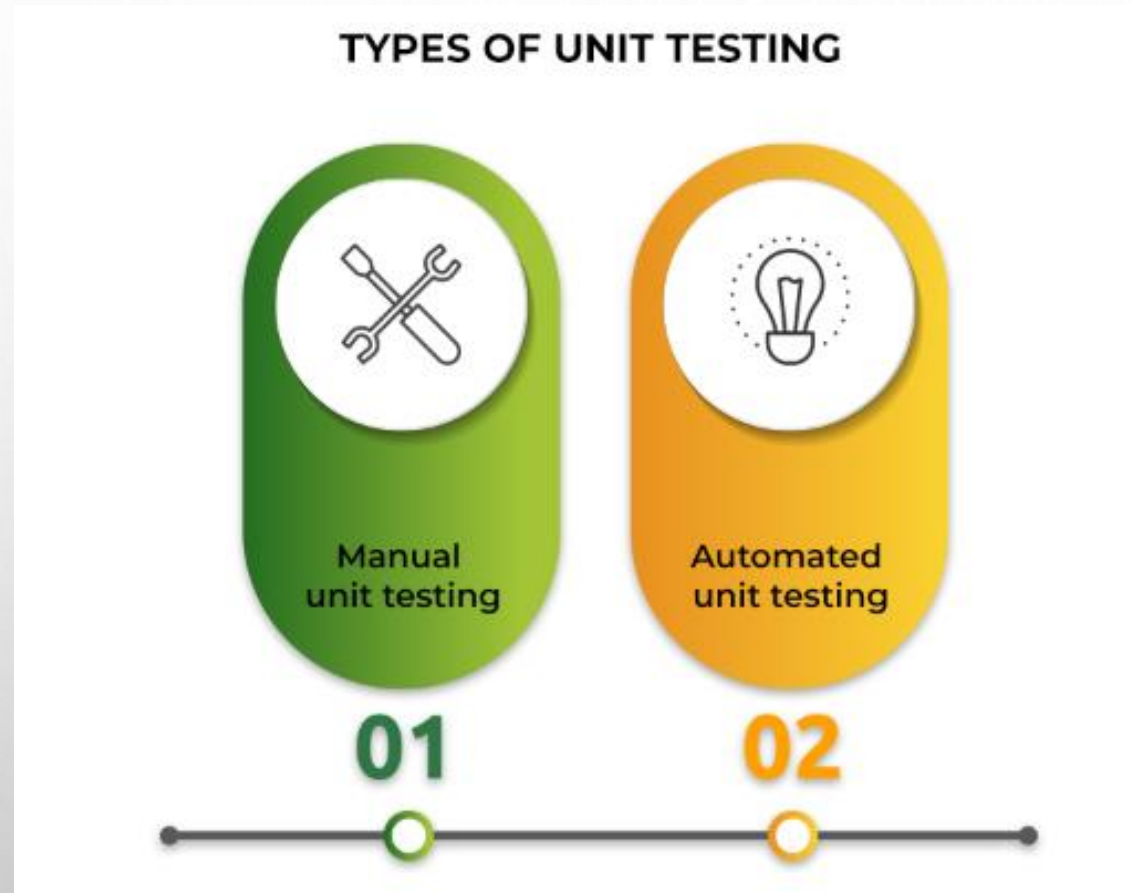
## Benefits of Unit Testing in SDLC



# WHY?

- Code covered with tests is more reliable than the code without.
- If a future change breaks something in the code, developers will be able to identify the root of the problem right away rather than coming through an unwieldy codebase to find the issue.
- Unit tests help developers detect problems immediately, then fix them quickly. With fewer resources spent finding bugs, teams can move on to the next phase of a project.
- **UNIT TESTING SAVES TIME AND MONEY:** Many bugs are found at the software construction stage, which prevents the transition of these bugs to the following stages, including after the release of the product. This saves the costs of fixing the bugs later in the development lifecycle and also brings benefits to end-users, who don't have to deal with a buggy product. Also, you will greatly benefit from improved [test time estimation](#), saving lots of time and resources.

# TYPES OF UNIT TESTING



# MANUAL UNIT TESTING?

- Manual testing is unit testing executed without special apps or programs. Every step of the testing process is carried out by individuals (developers or otherwise). Manual unit testing is not frequently seen due to better alternatives and multiple drawbacks. Manual unit testing is cost intensive as workers must be paid for the time expended during the process, especially when it involves non-permanent staff. The process is time-consuming as tests ought to be run each time the code is altered.
- It can also be challenging to isolate and test independent units, making discerning the source of faults elicited during the testing process very difficult. The developer frequently performs manual testing and assesses the software's stability after adding or removing lines of code.



# AUTOMATED UNIT TESTING

- Automated unit testing is a type of testing that is done without substantial human involvement. The tools used to execute automated tests are ultimately produced by people.
- 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.



# WORKFLOW AND CYCLE

1

- **Create Test Cases**

2

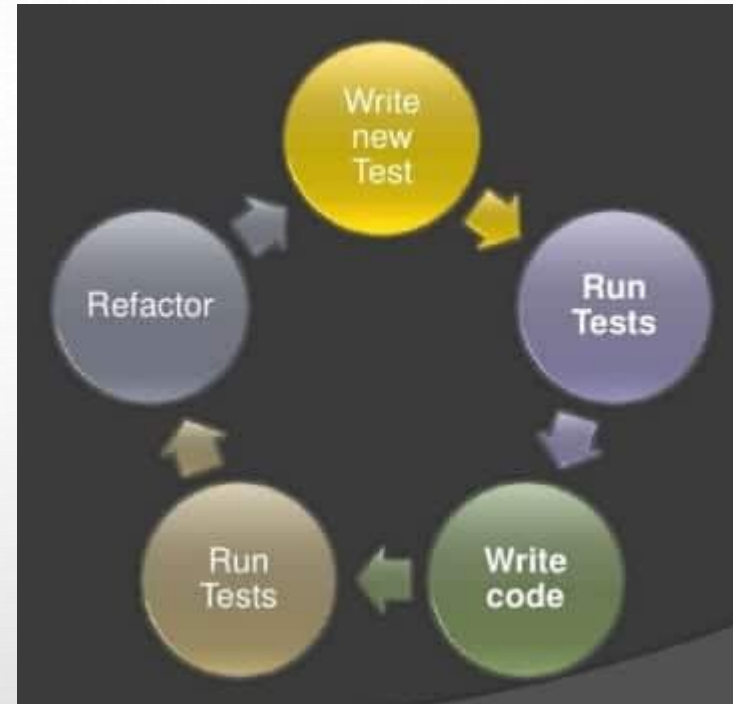
- **Review/Rework**

3

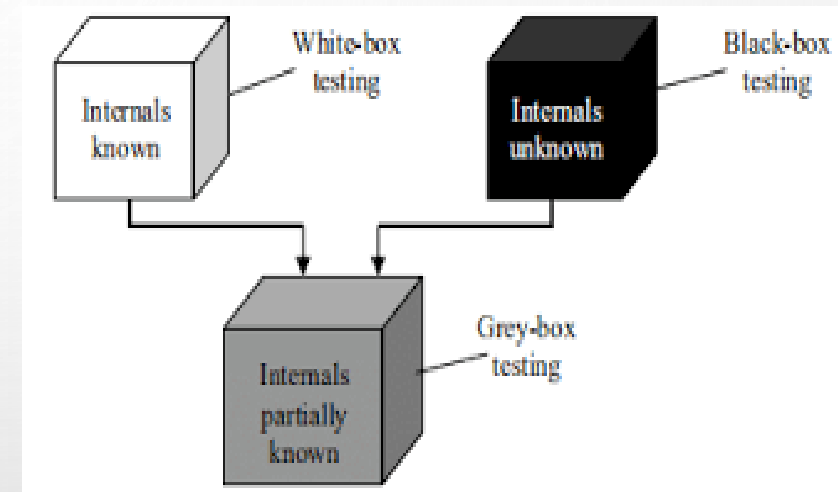
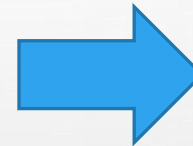
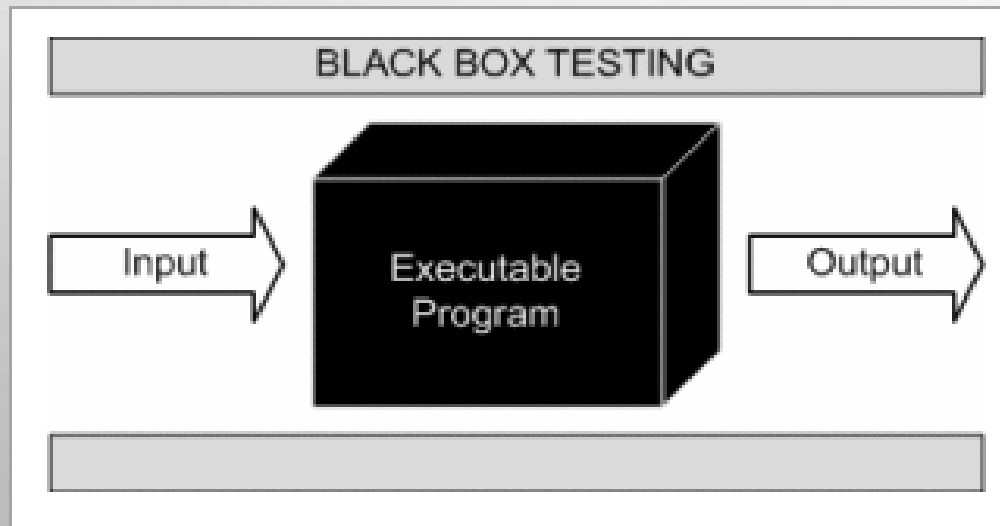
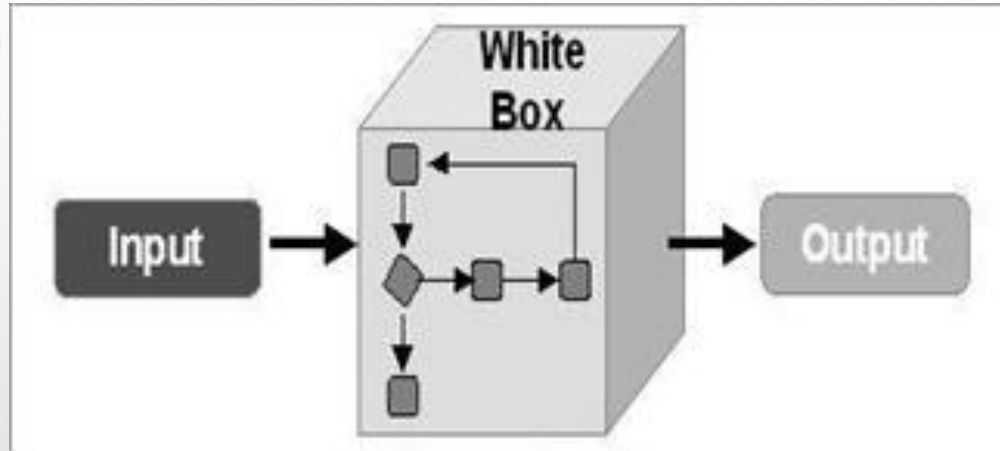
- **Baseline**

4

- **Execute Test Cases**



# UNIT TESTING TECHNIQUES



- Matrix Testing.
- Pattern Testing.
- Orthogonal Pattern Testing.
- Regression Testing.

# 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.

# WHITEBOX TESTING

- White box testing is a testing technique in which software's internal structure, design, and coding are tested to verify input-output flow and improve design, usability, and security. In white box testing, code is visible to testers, so it is also called clear box testing, open box testing, transparent box testing, code-based testing, and glass box testing.
- White box testing involves the testing of the software code for the following:
  - Internal security holes
  - Broken or poorly structured paths in the coding processes
  - The flow of specific inputs through the code
  - Expected output
  - The functionality of conditional loops
  - Testing of each statement, object, and function on an individual basis

# How do you perform White Box Testing?

- **UNDERSTAND THE SOURCE CODE**
- **CREATE TEST CASES AND EXECUTE**

```
Printme (int a, int b) {  
    int result = a+ b;  
    If (result> 0)  
        Print ("Positive", result)  
    Else  
        Print ("Negative", result)  
}
```

----- Printme is a function

----- End of the source code

# WHITEBOX TESTING TECHNIQUES

- STATEMENT COVERAGE
- DECISION COVERAGE
- BRANCH COVERAGE
- CONDITION COVERAGE
- PATH COVERAGE
- FINITE STATE MACHINE COVERAGE



# IMPLEMENT UNIT TESTING

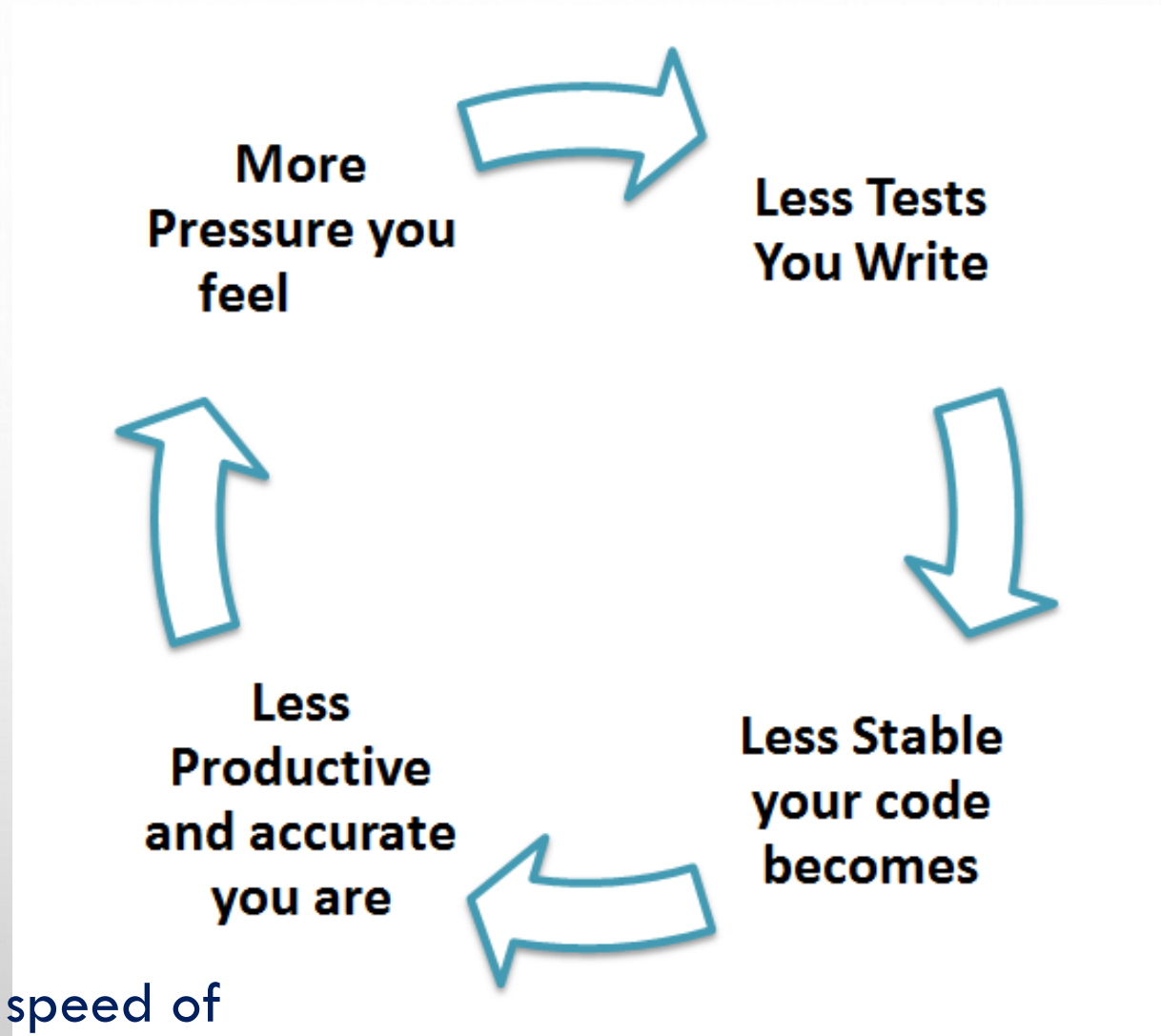
1. Write appropriate test names
2. Create simple tests
3. Craft deterministic tests
4. Address a single use-case
5. Aim for maximum test coverage
6. Design unit tests to be as fast as possible
7. Minimize test dependencies
8. Adopt test automation

# UNIT TESTING FRAMEWORK

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.

# Unit Testing Myth

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



Truth is Unit testing increase the speed of development.

# How to write good Unit Tests?

1. Unit tests should be written to verify a single unit of code and not the integration.
2. Small and isolated unit tests with clear naming would make it very easy to write and maintain.
3. 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.
4. It should run quickly.
5. Unit test should be reusable.

# How to Accept Unit Testing?

Good unit testing can be carried out in 3 basic parts.

1. Write a unit test code.

2. Run the unit test code to see if it meets the system requirements.

3. Execute the software code to test for any defects and see whether the code meets the system requirements.

AFTER UNDERTAKING THE ABOVE 3 STEPS, IF THE CODE APPEARS TO BE CORRECT THEN THE UNIT TEST IS SAID TO BE PASSED. IF IT DOES NOT MEET THE SYSTEM REQUIREMENTS, THEN THE TEST FAILS. IN THIS CASE, THE DEVELOPER NEEDS TO RECHECK AND CORRECT THE CODE.