Fpt University

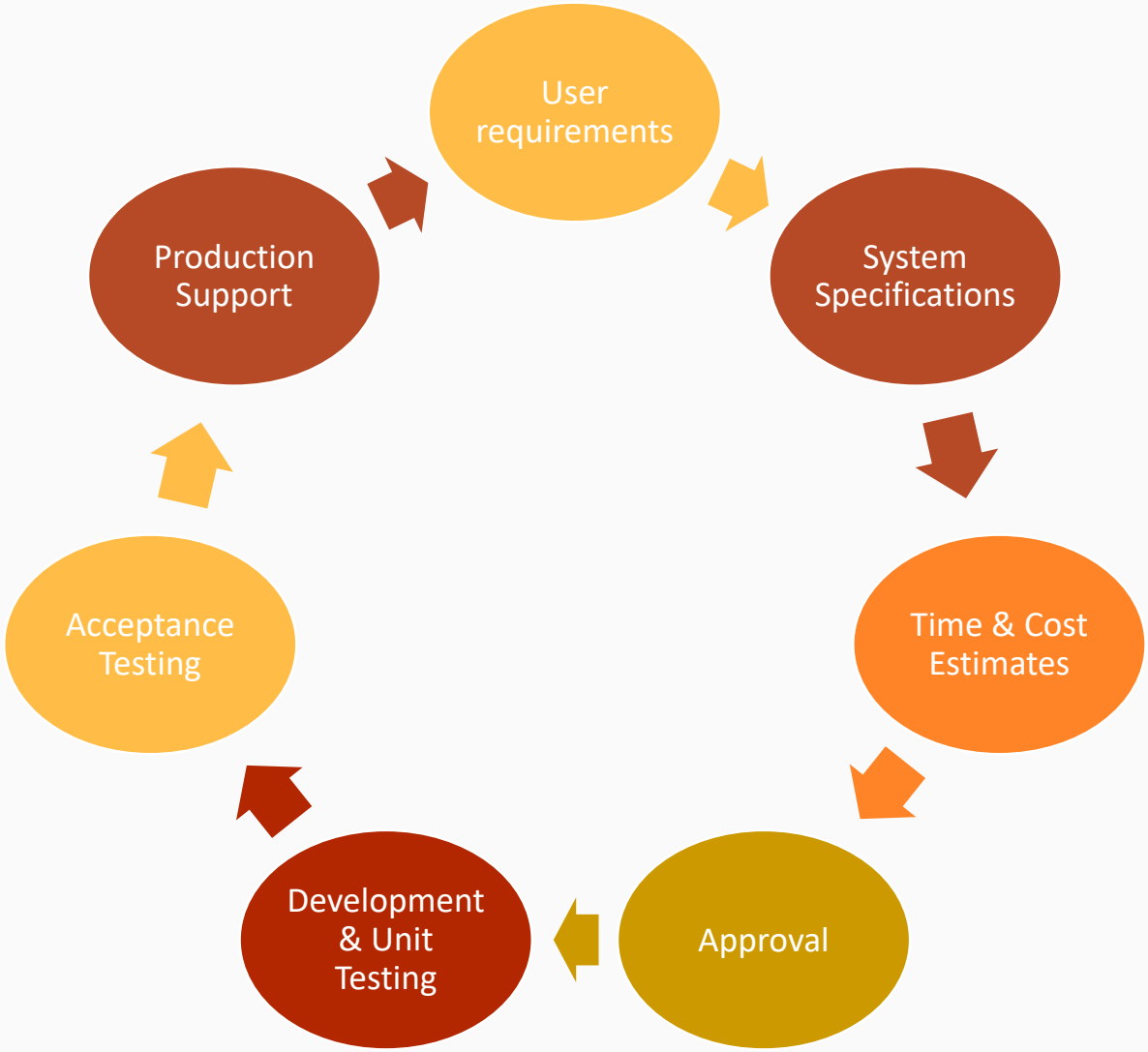
SWT301

LAB 2: Unit Testing

UT in SDLC

- Unit Testing is not a new concept. It's been there since the early days of programming.
- Unit tests to improve code quality by verifying every unit of the code used to implement functional requirements.
- Test driven development (TDD) or test-first development.
- Done by developers and sometimes White box testers

Unit Testing In SDLC



Unit Testing In SDLC

- Unit Testing is used to design robust software components that help maintain code and eliminate the issues in code units.
- Finding and fixing defects in the early stage of the software development cycle.
- It is a related part of the agile software development process.
- If we set this as a standard process, many defects would be caught in the early development cycle, saving much testing time.

Unit Testing In SDLC

Notes:

- Many developers hate to write unit tests. They either ignore or write bad unit test cases due to tight scheduled or lack of seriousness (yea they write empty unit tests, so 100% of them pass successfully.
- It’s important to write good unit tests or don’t write them at all.

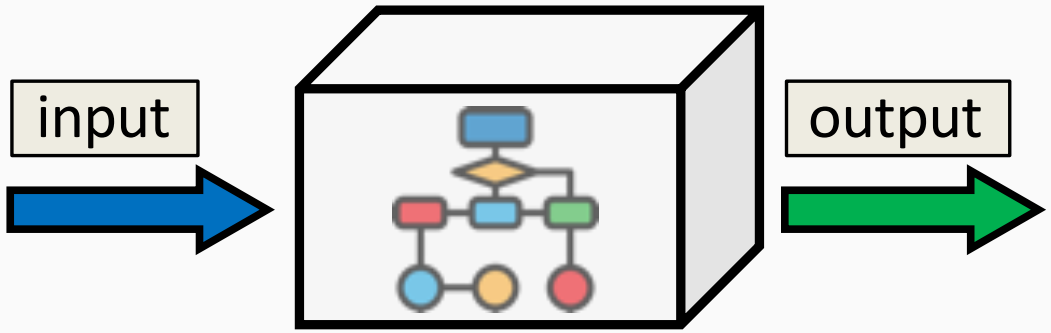
Unit Testing Methods

UT can be performed in 2 ways :

1. Manual Testing
2. Automated Testing

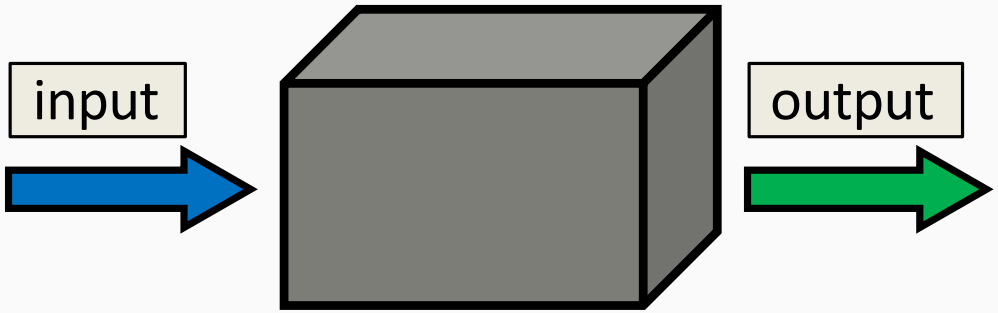
Techniques Within Unit Testing

- **White box testing**
  - Tester knows the internal structure of the software including the code
  - Tester can test it against the design and the requirements
  - White-box testing is also known as transparent testing.



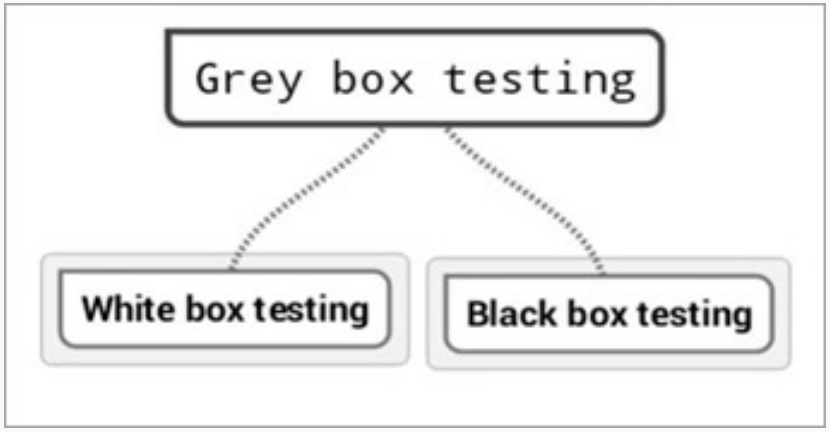
Techniques w/i Unit Testing

- **Black box testing**
  - Tester does not know the internal structures either the code of the software



Techniques w/i Unit Testing

- **Grey box testing**
  - The grey box is considered as a **combination of black box and white box testing techniques** (Semi-transparent technique testing)**.**
  - The testers are only partially aware of the internal structure, functions, and designs along with the requirements



Benefits of Unit Testing

1. The process becomes agile
2. Code quality improves
3. Detects bugs early
4. Easier changes and simplified integrations
5. Documentation availability
6. Easy debugging process
7. Lower cost
8. Code completeness can be demonstrated using unit tests
9. Saves development time
10. Code coverage can be measured

How to write good Unit Tests?

- A Unit test 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 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
- A Unit test should be reusable

Lab 2

- Tool: JUnit.
- Deliverables: (utilize the provided Unit Testing Template)
  - UT Test cases
  - Test Report
  - Source code + Test code
- Grade levels [LOC = Lines of (Source) Code]
  - No submission : 0
  - < 500 LOC : 1 – 4 <= **99**
  - $500 - 1000$ LOC : 5 **150**
  - $1001 - 1500$ LOC : 6 **200**
  - $1501 - 2000$ LOC : 7 **250**
  - > 2000 LOC : 8 **300**
  - Complexity of UT : $+0 - 2$
  - 1 – 24: 1; 25 – 49: 2; 50 – 74: 3; 75 – 99: 4

Popular Unit Testing Tools

Guide (Vietnamese)

Giới thiệu JUnit - GP Coder (Lập trình Java)

#1) NUnit

- NUnit is a unit testing framework based on.NET platform
- It is a free tool allows to write test scripts manually but not automatically
- NUnit works in the same way as JUnit works for Java
- Supports data-driven tests that can run in parallel
- Uses Console Runner to load and execute tests

#18) JUnit

- JUnit is an open-source unit testing framework designed for Java Programming Language
- Supportive for the test-driven environment and the core idea on which it is based is ‘first testing than coding’
- Test data is first tested and then inserted in the piece of code
- Provides annotation for test method identification, an assertion for testing expected results and test runners
- Simplest and helps to write code easily and faster