HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Software Quality Assurance
Đảm bảo chất lượng phần mềm

Lecture 3: Introduction to Software Testing

Outline

- Definition and Motivation of Software Testing
- Basic terms of Software Testing
- Testing Activities
- Testing Levels

3.1. Definitions & Motivation

What is software testing?

- IEEE defines software testing as:
  - "A process of analyzing a software item to detect the **difference between existing and required conditions** (defects/errors/bugs) and to evaluate the features of the software item"
- Note that…
  - …one needs to know the **required conditions**
  - …one needs to be able to observe the **existing conditions**

Software Testing is a process

- Evaluating a program or a system
  - Verify that the product is right
  - Validate that the right product is developed
  - Evaluate the product quality
  - Find defects

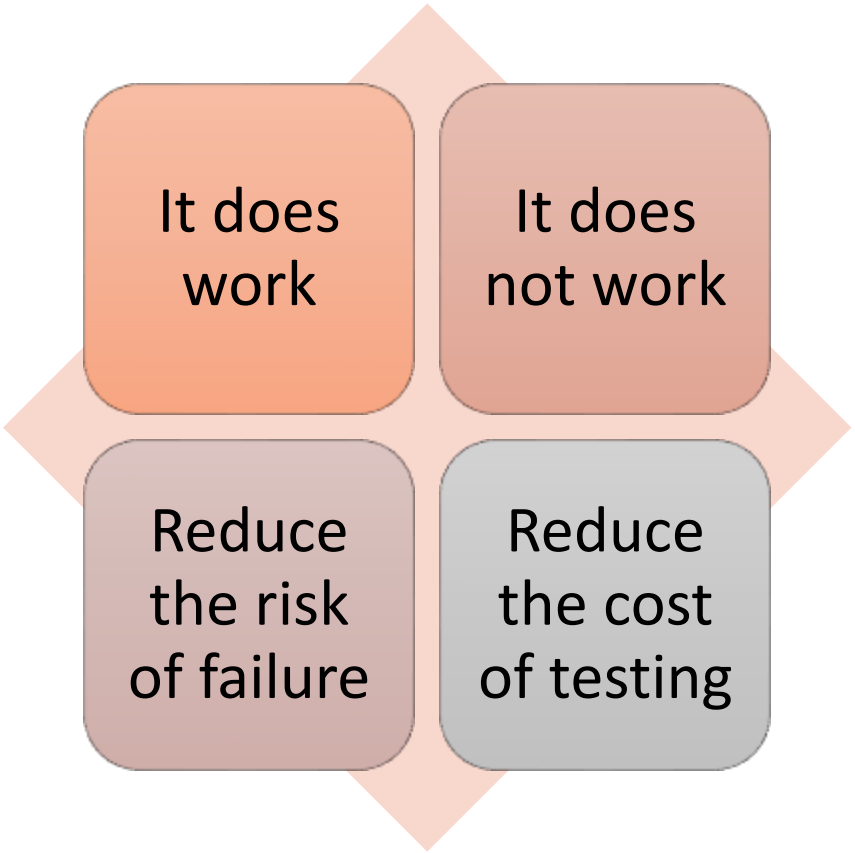
Testing takes time

"Reports estimate that regression testing consumes as much as 80% of the testing budget and can consume up to 50% of the cost of the software maintenance"

–Chittimalli and Harrold, 2009

Chittimalli, P, & Harrold, M 2009, 'Recomputing coverage information to assist regression testing', IEEE Transactions On Software Engineering, 35, 4, pp. 452- 469

Objectives of Testing



3.2. Basic terms of Software Testing

Test case
Trường hợp kiểm thử/Ca kiểm thử

- Simply, a pair of **<input, expected outcome>**
- Example: a square root function of nonnegative numbers
- Two kinds of testing function
  - stateless function/system
    - example: compiler
  - state-oriented system
    - example: ATM machine to withdraw function

TB_1: < 0, 0 >,
TB_2: < 25, 5 >,
TB_3: < 40, 6.3245553 >,
TB_4: < 100.5, 10.024968 >.

TS_1: < check balance, $500.00 >, < withdraw, "amount?" >, < $200.00, "$200.00" >, < check balance, $300.00 > .

Expected Outcome

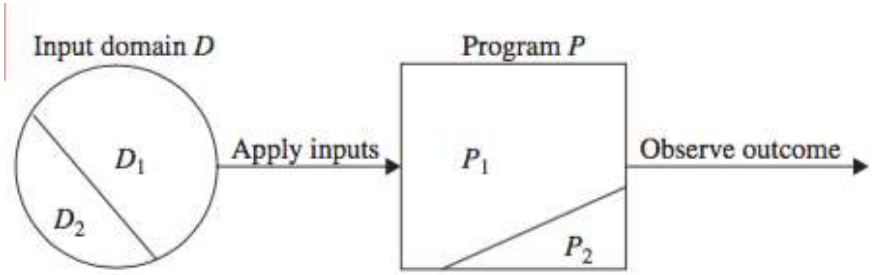
- An **outcome** of program execution may include
  - Single values produced by the program
  - State changes
  - Sequence or set of values which must be interpreted together for the outcome to be validated
- The concept of **oracle** in test designing: any entity (program, process, body of data…) producing the expected outcome of a particular test
- Expected outcome should be computed when designing test case
  - Without executing program but using program's requirements/specifications

Complete Testing
Hoàn tất kiểm thử

- Complete Testing means:
  - There are no **undiscovered** faults at the end of the test phase
- It is impossible to complete testing for most of the software systems
  - **Domain** of possible inputs of a program is too large to be completely used in testing
    - **valid** and **invalid** values of data input
  - **Design issues** may be too complex to completely test
  - Impossible to create all possible **execution environment**s of the system

Centre Issue in Testing
Vấn đề trung tâm của kiểm thử

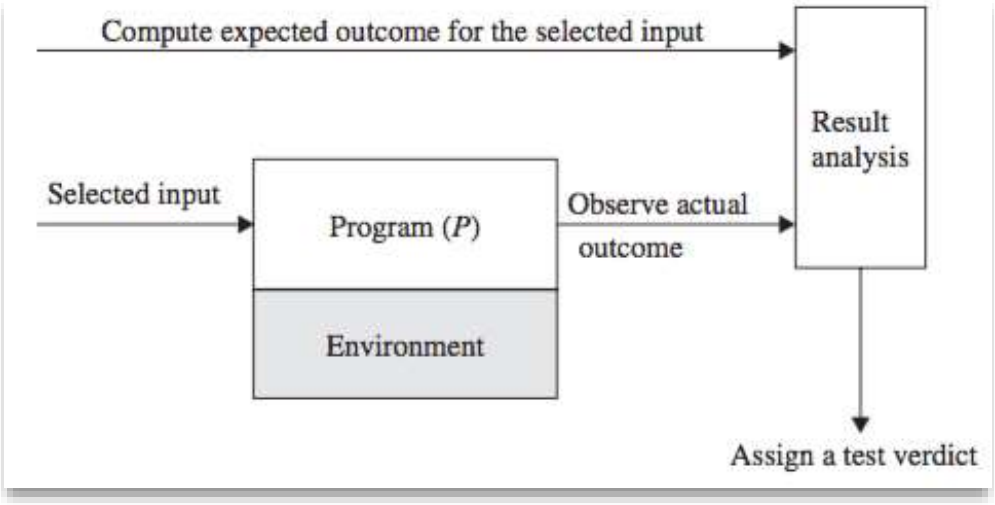
- Select a subset of the input domain to test a program
- Let D be the input of the program P
  - Select subset D1 ⊂ D to test P: D1 exercise only in part P1 ⊂ P (the execution behavior of P)
  - We said that we were attempting to deduce properties of P on selected inputs D1



3.3. Testing Activities

Testing Activities
Các hoạt động kiểm thử

- Identifying an objective to be tested
- Select inputs
- Compute the expected outcome
- Setup the execution environment of the program
- Execute the program
- Analyze the test result
- Writing test report if test case is failed



Sources of Information for Test Case
Nguồn dữ liệu sử dụng để thiết kế ca kiểm thử

- Objective of designing test cases: generate **effective** tests at a **lower cost**
- Sources of Information:
  - Requirements and functional specifications
  - Source code
  - Input and Output domains
  - Operational Profile
  - Fault Model

Requirements and Functional Specifications

- The process of software development begins by capturing user needs
- The number of user needs identified at the beginning depends on the specific life-cycle model
  - ex: waterfall vs agile
- Test case designing is performed based on software requirements
- Requirements can be specified in different manners
  - Plaintext, equations, figures, flowcharts…
  - Use cases, entity-relationship diagrams, class diagrams
  - Using formal language and notation as finite-state machine…

Source Code

- Requirement specifications describe the intended behavior of a system
- Source Code describes the actual behavior of the system
- Test cases must be designed based on the program

Input and Output domains

- Input domain
  - Valid data and Invalid data
- Some values in the input domain may have special meanings and should be treated separately
- Special values in the input and output domains of a program should be considered while testing the program

Operational Profile

- **Operation/Usage profile** is a quantitative characterization of how a **system will be used**
- It is important to test a system by considering the ways it will actually be used
- To guide test engineers in selecting test cases using samples of system usage
- Test inputs are assigned a probability distribution or profile according to their occurrences in actual operation

Fault Model

- Encountered faults are a excellent source of information in designing new test cases
- Different classes of faults: initialization faults, logic faults, interface faults
- 3 types of fault-based testing:
  - Error guessing
  - Fault seeding
  - Mutation testing

Test case designing techniques

- White Box Testing
  - Structural testing techniques
  - Examining source code with a focus on control flow and data flow
  - Applied to individual units of a program
- Black Box Testing
  - Functional testing techniques
  - Examining functional specifications of testing modules
  - Applied to both an entire system and the individual program units
- These strategies are used by different groups of people at different times during a software life cycle
  - Ex: programmers use both to test their own code while quality assurance engineers apply the idea of functional testing

Test Planning and Design

- The purpose of system test planning is to get ready and organized for test execution
- Test plan provides a framework, scope, details of resource needed, effort required, schedule of activities and a budget
- Test design is a critical phase of software testing
  - System requirements are critically studied
  - System featured to be tested are identified
  - Objectives and detailed behaviors of test cases are defined
- Test-Driven Development (TDD): test cases are designed and implemented before the production code is written
  - key practice of modern agile software development processes

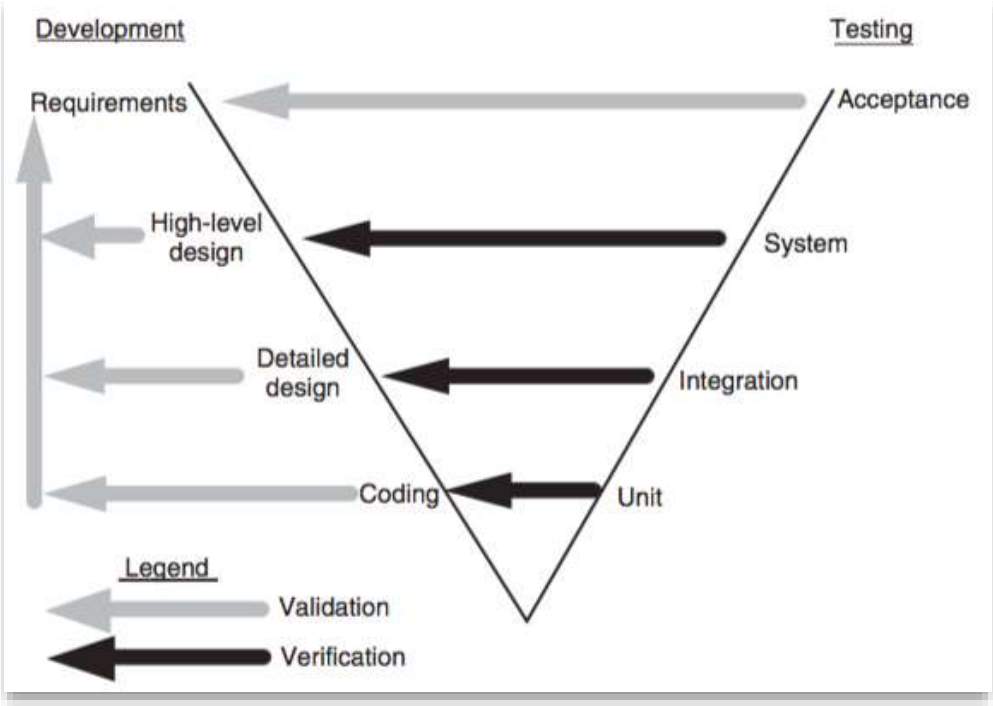
Monitoring and Measuring test execution

- Motivation: to explore the progress of testing and reveal the quality level of the system
- Test execution metrics are categorized into 2 classes
  - Metrics for monitoring test execution
  - Metrics for monitoring defects
- Quantitative measurements
  - Number of defects detected by test cases
  - Number of test cases designed/executed per day
  - Number of defects found by the customers that were not found by the test engineers

3.4. Testing Levels

Testing Levels
Mức độ kiểm thử

- Testing is performed at different levels involving the whole system or parts of it throughout the life cycle of a software product
- Four stages of testing
  - Unit test
  - Integration test
  - System test
  - Acceptance test

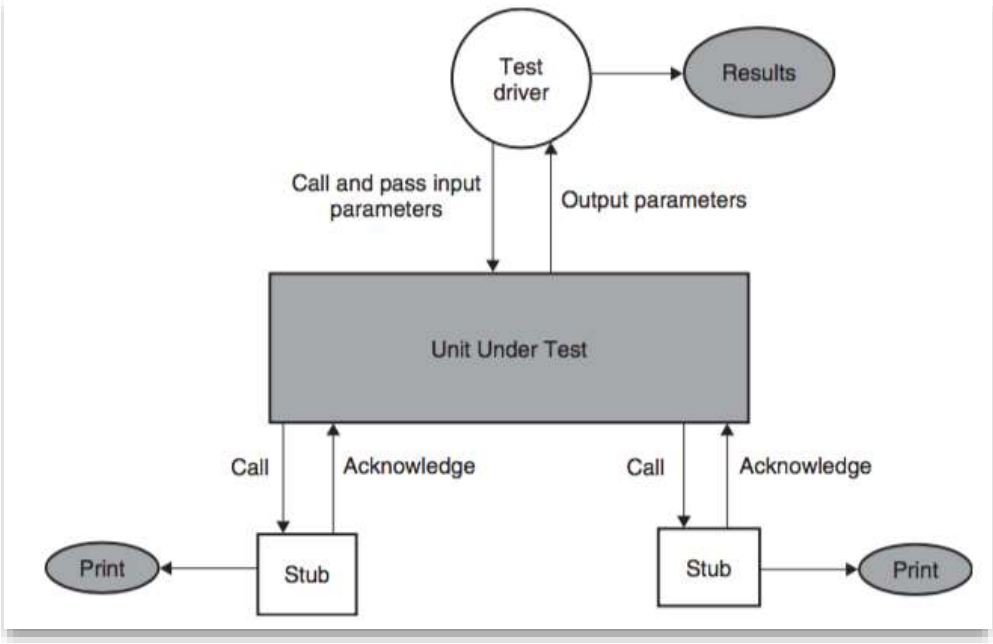


Unit Testing
Kiểm thử đơn vị

- Unit = smallest testable software component
  - Objects
  - Procedures/Functions
  - Reusable components
- Tested in isolation
- Usually done by programmer during development
  - A pair tester can help
  - Developer guide or coding guideline can require
- Also known as component or module testing

Dynamic Unit Test Environment

- An environment for dynamic unit testing is created by emulating the context of the unit under test
- The context consists of:
  - A caller of the unit
  - All the other units called by the unit



Test Driver

- The caller unit is called ***test driver***
  - A program that invokes the unit under test
  - The unit under test executes with input values received from the driver and return a value to the driver
  - The driver compares the actual outcome with the expected outcome and reports the ensuing test result

Stubs

- The units called by the unit under test are called ***stubs***
  - A stub is a *dummy subprogram* that replaces a unit that is called by the unit under test
- A stub performs two tasks:
  - It shows an evidence that the stub was called
  - It returns a precomputed value to the caller so that the unit under test can continue its execution

Integration Testing
Kiểm thử tích hợp

- At the unit testing level, the system exists in pieces under the control of the programmers
- Put the modules together to construct the complete system
- The path from tested components to constructing a deliverable system contains two major testing phases:
  - **Integration testing**
  - **System testing (introduced in Lecture 6)**

Motivation of Integration Testing

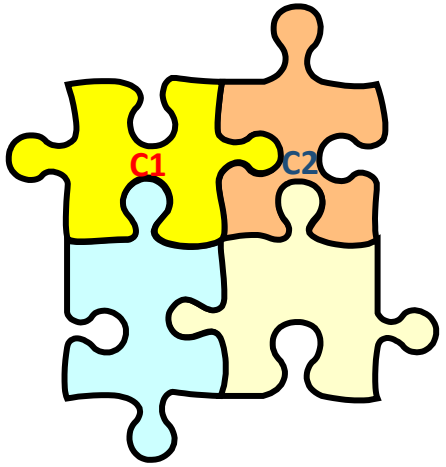
- Different modules are generally created by groups of different developers
- Interface errors between modules created by different programmers and even by the same programmers are rampant
- Unit testing of individual modules is carried out in a controlled environ- ment by using test drivers and stubs. It is therefore difficult to predict the behavior of a module in its actual environment after the unit testing is performed
- Some modules are more error prone than other modules, because of their inherent complexity. It is essential to identify the ones causing most failures.

Objectives of Integration Testing

- To build a WORKING version of the system by putting the modules together
- To ensure that additional modules work as expected without disturbing the functionalities of the modules already put together
- Integration test is completed when the system is fully integrated together:
  - All the test cases have been executed
  - All the severe and moderate defects found have been fixed and the system is retested

Interfaces between modules

- Modules are interfaced with other modules to realize the system's functional requirements
- An interface between two modules allows one module to access the service provided by the other
- An interface between two modules implements a mechanism for passing control and data between modules



Different types of Interfaces between modules

Three common paradigms for interfacing modules

- **Procedure Call Interface**: a procedure in one module call a procedure in another module. The caller passes on control and data to the called module. The called one can pass data to the caller while returning control back to the caller
- **Shared Memory Interface**: A block of memory is shared between two modules. Data are written into the memory block by one module and are read from the block by the other
- **Message Passing Interface**: One module prepares a message by initializing the fields of a data structure and sending the message to another module. This form is common in client-server or web-based system

Interface Errors

If all the unit tested modules work individually, why can these modules not work when put together?

Interface Errors

- **Construction**: interface specification is separated from the implementation code (i.e., in C/C++ through #include header.h). Inappropriate use of #include statements cause construction errors
- **Inadequate Functionality**: Implicit assumptions in one part of a system that another part of the system would perform a function
- **Location of Functionality**: Disagreement on or misunderstanding about the location of a functional capability within the software leads to this sort of error
- **Changes in Functionality**: Changing one module without correctly adjusting for that change in other related modules affects the functionality of the program

Interface Errors (cont.)

- **Added Functionality**: A completely new functional module was added as a system modification
- **Misuse of Interface**: one module makes an error in using the interface of a called module. Interface misuse can take the form of wrong parameter type, wrong parameter order or wrong number of parameters passed
- **Misunderstanding of Interface**: A calling module may misunderstand the interface specification of a called module. The called module may assume that some parameters passed to it satisfy a certain condition whereas the caller does not ensure that the condition holds.
- ...

Advantages of Integration Testing

- Defects are detected early
- It is easier to fix defects detected early
- Get earlier feedback on the health and acceptability of the individual modules and on the overall system
- Scheduling of defect fixes is flexible and it can overlap with development

Who do integration testing?

- System integration testing is performed by the system integration group, aka a build engineering group
  - Integration test engineers who should be familiar with the interface mechanisms
  - Team of engineers who built the modules
  - The system architects because of the fact that they have a bigger picture of the system

Granularity of System Integration Testing

- System integration testing is performed at different levels of granularity
- Use both of two techniques
  - Black box testing
  - White box testing
- Different levels:
  - Intrasystem Testing
  - Intersystem Testing
  - Pairwise Testing

Intrasystem Testing

- Low-level integration testing with the objective of combining the modules together to build a cohesive system
- Combining modules progresses in an incremental manner
- Example: in a client-server system, individual modules of client and server are combined in an incremental fashion, to be tested
- Source of testing: low-level design document which details the specification of modules

Intersystem Testing

- A high-level testing phase which requires interfacing independently tested systems
- All systems are connected together
- Testing is conducted from end to end: ensuring that the interaction between systems work together but not to conduct a comprehensive test
- Only one feature is tested at a time and on a limited basis
- Test cases are derived from the high-level design document, which details the overall system architecture
- Ex: integrating client and server to test after perform individually tested system (client and server)

Pairwise Testing

- There can be many intermediate levels of system integration testing between intrasystem testing and intersystem testing
- Pairwise testing is a kind of intermediate level of integration testing
- Only two interconnected systems in an overall system are tested at a time
- The purpose is to ensure that two systems under consideration can function together
- All other systems are assumed to behave as expected

System Integration Testing Techniques
Các kĩ thuật kiểm thử tích hợp hệ thống

- Incremental
- Top-down
- Bottom-up
- Sandwich
- Big bang

Incremental technique

- Integration testing is conducted in an incremental manner as a series of test cycles
- In each test cycle: a few more modules are integrated with an existing and tested build to generate a larger build
- Objective:
  - Complete one cycle of testing
  - Fix all the errors found
  - Continue the next cycle of testing
  - Finish building a complete and ready system for system testing

How many test cycles for system integration testing?

The number of system integration test cycles and the total integration time depend on:

- Number of modules in the system
- Relative complexity of the modules (McCabe Cyclomatic Complexity)
- Relative complexity of the interfaces between modules
- Number of modules needed to be clustered together in each test cycle
- Whether the modules to be integrated have been adequately tested before
- Turnaround time for each test-debug-fix cycle

Constructing a build in each test cycle

- Constructing a build is a process by which individual modules are integrated to form an interim **software image**
- **A software image** is a compiled software binary
- The **final build** will be a candidate for system testing
- Creating a daily build is very popular to facilitate a faster delivery of the system
- Focusing on small incremental testing, steadily increasing the number of test cases and regression testing from build to build

Advanced Testing Level

- System Testing
- Acceptance Testing
- Regression Testing

Summary

- Students should understand principle terms of software testing
- Have a general understand of V-model
- Understand different phase/activities of testing process

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Thank you for your attention!!!