

COURSE NAME

SOFTWARE QUALITY  
AND TESTING

CSC 4133

(UNDERGRADUATE)

---

## CHAPTER 6

UNIT TESTING

---

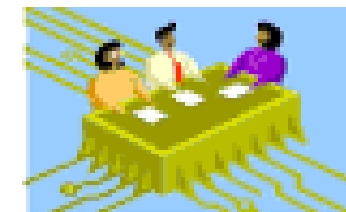
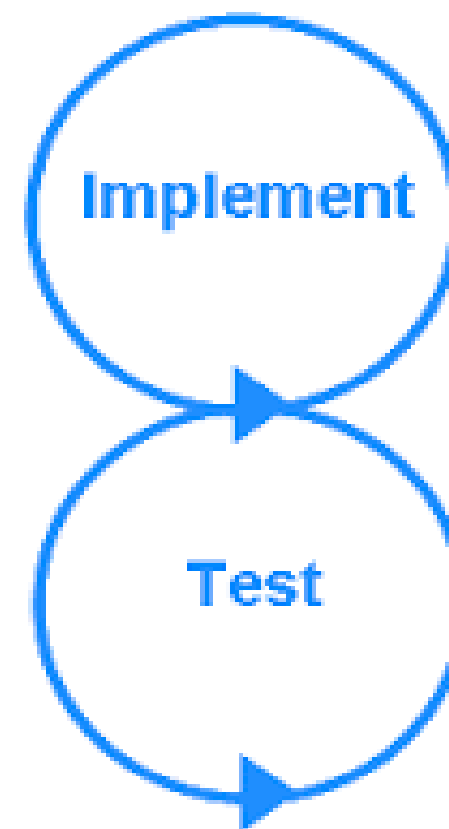
## UNIT TESTING

- Tests a **small software unit** at a time, which is typically performed by the **individual programmer** who implemented the unit **prior to Integration testing**
- **Unit Testing** is usually performed by using the **White Box Testing** method
- **Lower level/ initial level testing**
- ❑ **Static Unit Testing**
- ❑ **Dynamic Unit Testing**
- ❖ Static and Dynamic analysis are **complementary** in nature
- ❖ It is recommended that **static unit testing** be **performed prior** to the **dynamic unit testing**



## STATIC UNIT TESTING (REVIEW CODE)

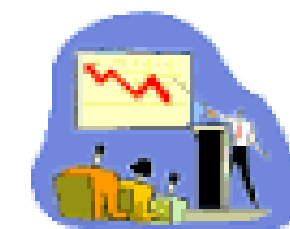
- Code is examined at **compile time** over all possible behaviors that might arise during run time
- Code of each unit is validated against requirements of the unit by reviewing the code
- In static unit testing, code is reviewed by applying techniques:
  - Walkthrough (informal)
  - Code Inspection (formal)
- Black-box and White-box testing



Walkthroughs



Inspections




Reviews

## DYNAMIC UNIT TESTING (EXECUTE CODE)

- Execute at **Run time** testing of Code
- Black-box testing

```
public string BookActor(string theActor)
{
    string details = "Bookin' change if"
    " actor starts tr
    string theActor = "Ac
    if (theDate != string.
    {
        return theActor + eDate +
```



## STEPS IN THE CODE REVIEW PROCESS

**Step 1: Readiness** – The **author** of the code unit ensures that the unit under test is ready for review. A unit is said to be ready if it satisfies the following criteria – **Completeness, Minimal functionality, Readability, Complexity, Requirements and Design documents**

**Step 2: Preparation** – Each **reviewer** develops the following: **List of questions, Potential Change Request (CR), Suggested improvement opportunities**

**Step 3: Examination** – the examination process consists of the following:

- The **author** makes a presentation
- The **reviewer** reads the code
- The **record keeper** documents the CR
- **Moderator** ensures the review is on track

**CR (Change Request) includes the following details:**

- Give a brief description of the issue
- Assign a priority level (major or minor) to a **CR**
- Assign a person to follow it up
- Set a deadline for addressing a **CR**

## STEPS IN THE CODE REVIEW PROCESS

**Step 4: Re-work** – At the end of the meeting, the **record keeper** produces a summary of the meeting that includes the following information:

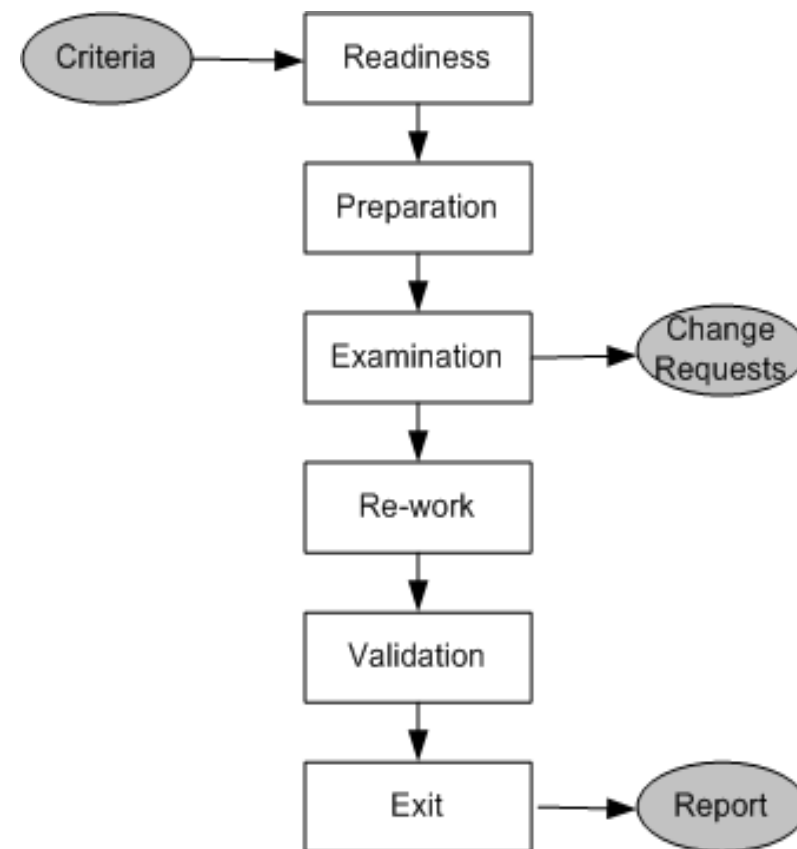
- Make the list of all the CRs
- Make a list of improvements/maintenance (**corrective, perfective, adaptive, preventive** maintenance)
- Record the minutes of the meeting (optional)
- Author works on the CRs to fix the issue

### Step 5: Validation

- CRs are independently validated (e.g. **regression testing**)

### Step 6: Exit

- A summary report of the meeting minutes is distributed among the stakeholder who have requested and who will be affected by the changes



## CODE REVIEW METRICS

### ❑ Identify the Cost of Development and Change

- Cost may calculate by How many lines the developer can write in each hour and How much money to pay him for each hour that finds out the cost of each line of code
- The number of lines of code (LOC) reviewed per hour
- The number of CRs generated per thousand lines of code (KLOC)

### ❑ Identify the Cost of Testing process

- Number of CRs generated per hour
- The total number of hours spend on code review process

## DYNAMIC UNIT TESTING

- Execution-based unit testing is referred to as dynamic unit testing
- A program unit is executed in isolation, and the outcomes of program execution are observed
- The environment of a unit is emulated and tested in isolation
- IDE checks every statement in the program for errors while writing
- The caller unit is known as test driver
  - A test driver is a program that invokes the unit under test (UUT)
  - It provides input data to UUT and report the test result
- The emulation of the units called by the UUT are called stubs
  - It is a “sub-program” that replaces a unit that is called by the UUT
- The test driver and the stubs are together called scaffolding
- The low-level design document provides guidance for selection of input test data

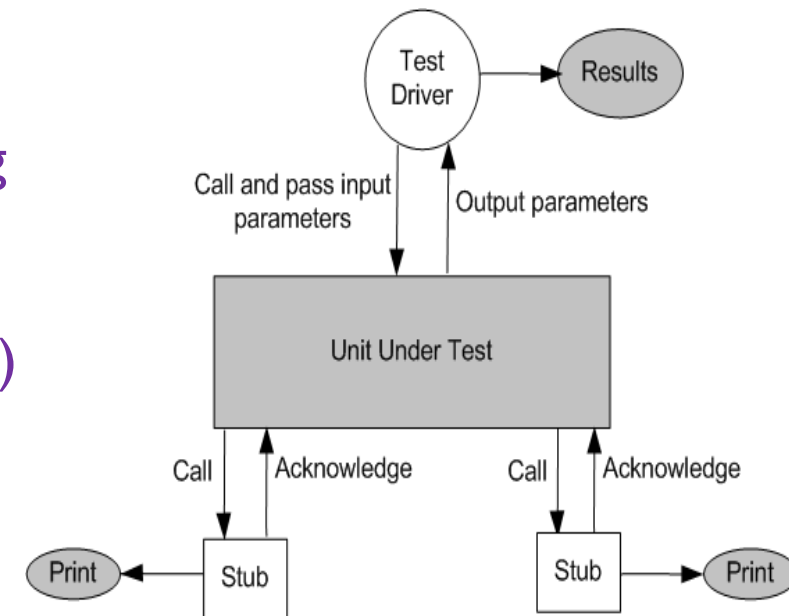


Figure : Dynamic unit test environment



## DYNAMIC UNIT TESTING

### ❑ Control flow testing

- Draw a control flow graph (CFG) from a program unit and Select a few control flow testing criteria
- Identify a path in the CFG to satisfy the selection criteria
- Derive the path predicate expression from the selection paths and one can generate the data

### ❑ Data flow testing

- Draw a data flow graph (DFG) from a program unit and then follow the procedure described in control flow testing (e.g. marks conversion and grading in VUES)

### ❑ Domain testing

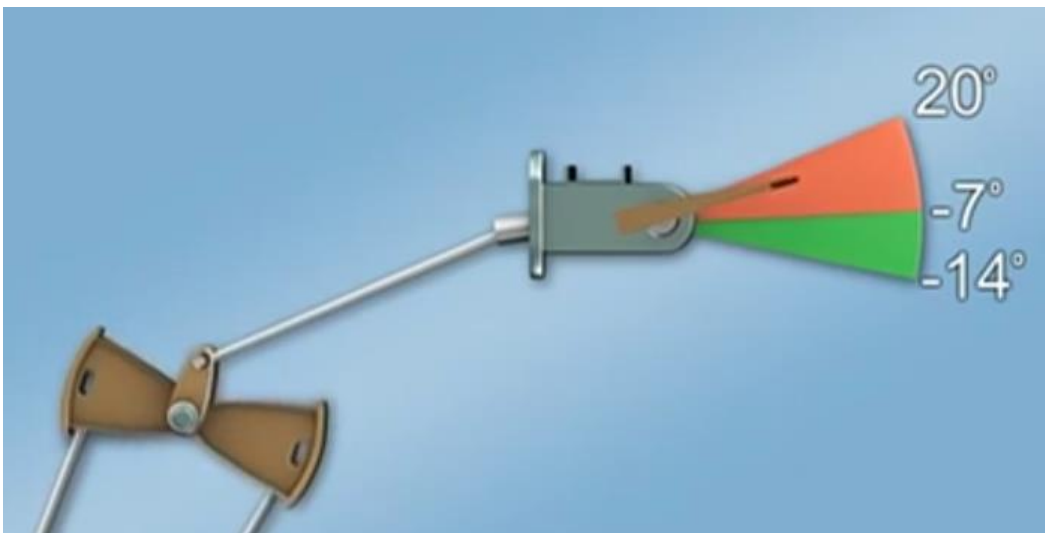
- A domain error occurs when a **specific input data** causes the program to **execute a wrong path** in the program (e.g. Leave application: for 1-5 days recommendation by team lead, 5-10 by PM)

### ❑ Functional program testing

- Input/output domains are defined to compute the input values that will cause the unit to **produce expected output values** (e.g. grading in VUES A+ for 90 marks)

## DEBUGGING

Debugging Identify the error and potential cause of error (hypothetical hidden errors)



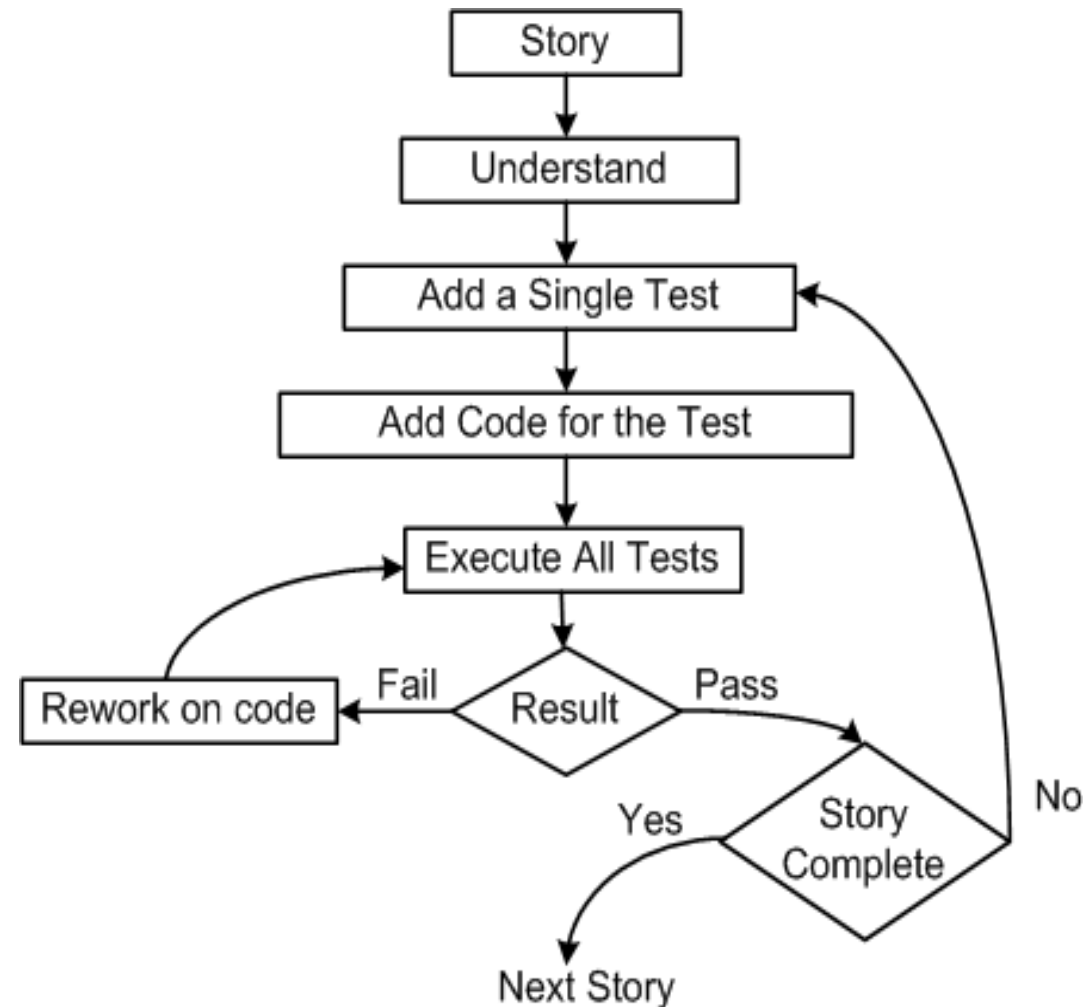
## DEBUGGING TECHNIQUES

- ❑ Identify the error and potential cause of error (hypothetical hidden errors)
- ❑ **Brute force testing**
  - most common; but least efficient
  - memory dumps are taken, run-time traces are invoked, and the program is loaded with output statements
- ❑ **Backtracking**
  - common debugging approach that can be used successfully in small programs
  - source code is traced backward (manually) until the cause is found
- ❑ **Cause elimination**
  - a “cause hypothesis” is devised
  - if initial tests indicate that a particular cause hypothesis shows promise, data are refined in an attempt to isolate the bug (c/a-b where the possibility of a-b is zero)

# UNIT TESTING IN EXTREME PROGRAMMING

## ❑ Pair programming in XP

1. Pick a requirement (i.e. story)
2. Write a test case that will verify a small part of the story and assign a fail verdict to it
3. Write the code that implement particular part of the story to pass the test
4. Execute all tests
5. Rework on the code, and test the code until all tests pass
6. Repeat step 2 to step 5 until the story is fully implemented



# UNIT TESTING

- **A mutation of a program** is a modification of the program created by introducing a single, small, legal syntactic change in the code.
- **Junit** is a framework for performing unit testing of Java programs.

## TOOLS FOR UNIT TESTING

- Code auditor
- Documenters
- Interactive debuggers
- Static code (path) analyzer
- Software inspection support
- Test coverage analyzer
- Test data generator
- Test harness
- Performance monitors
- Network analyzers
- Simulators and emulators
- Version control

## REFERENCES

- ❑ Software Testing And Quality Assurance – Theory and Practice - Kshirasagar Naik & Priyadarshi Tripathy
- ❑ Software Quality Engineering: Testing, Quality Assurance and Quantifiable Improvement - Jeff Tian