# Object Oriented Testing

A · P · U
ASIA PACIFIC UNIVERSITY
OF TECHNOLOGY & INNOVATION

## Lecture - 9

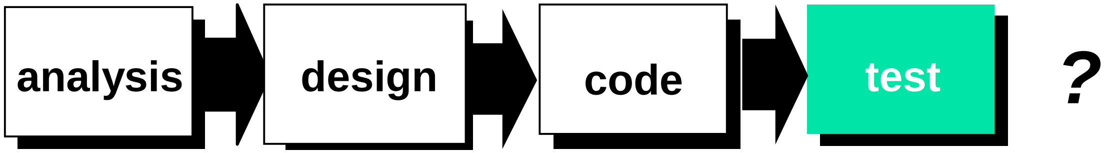
Source :”Applying UML patterns “,Craig Larman, second edition chapter 21

# Topics

- **Analysis and Design Testing**
- **Class Tests**
- **Integration Tests**
- **Validation Tests**
- **System Tests**

# Objectives

- To discuss when testing takes place in the life cycle



  - Test-driven development advocates early testing!
- To cover the strategies and tools associated with object oriented testing
  - Analysis and Design Testing
  - Class Tests
  - Integration Tests
  - Validation Tests
  - System Tests
- To discuss test plans and execution for projects

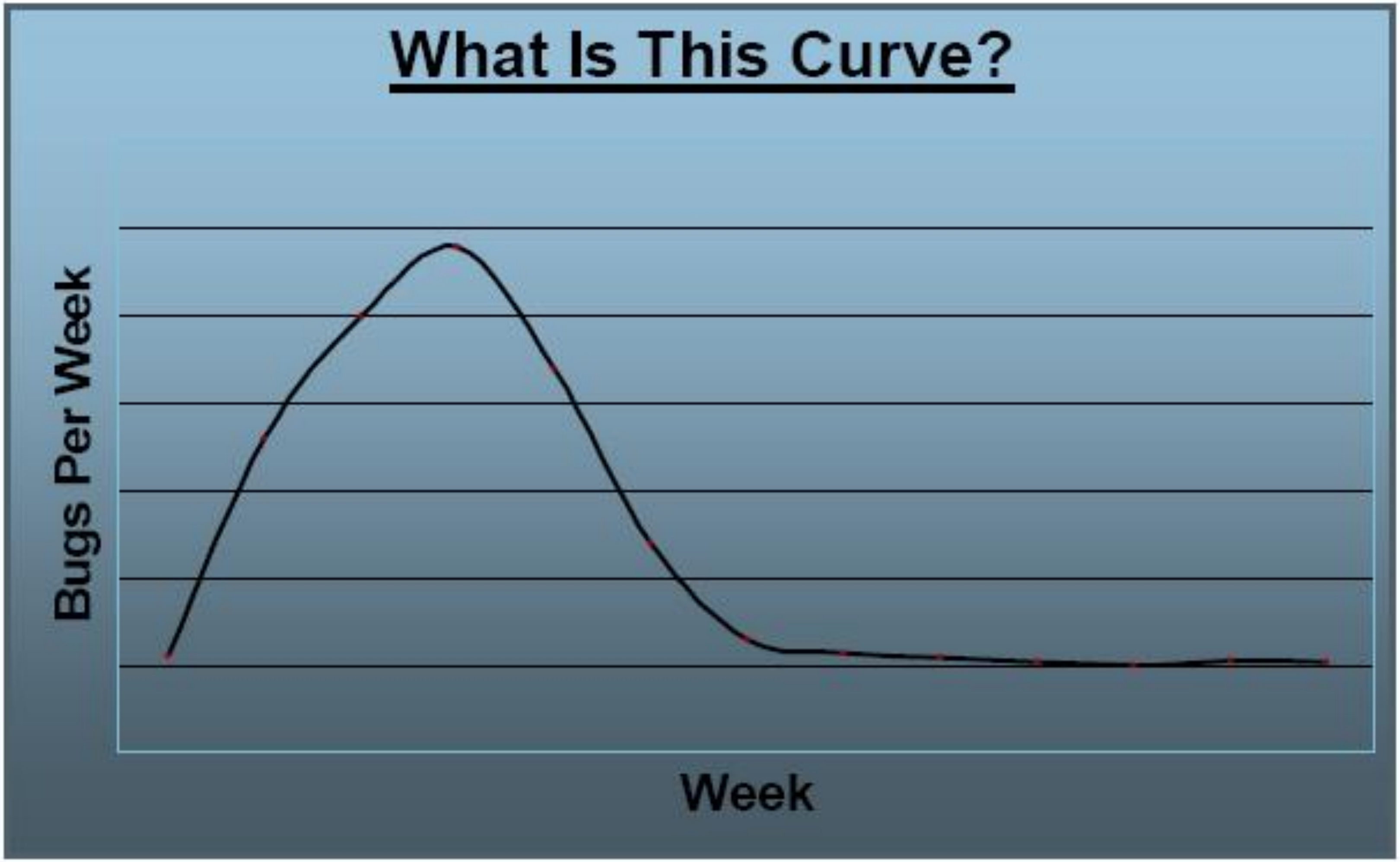
# Object-Oriented Testing

- *When should testing begin?*

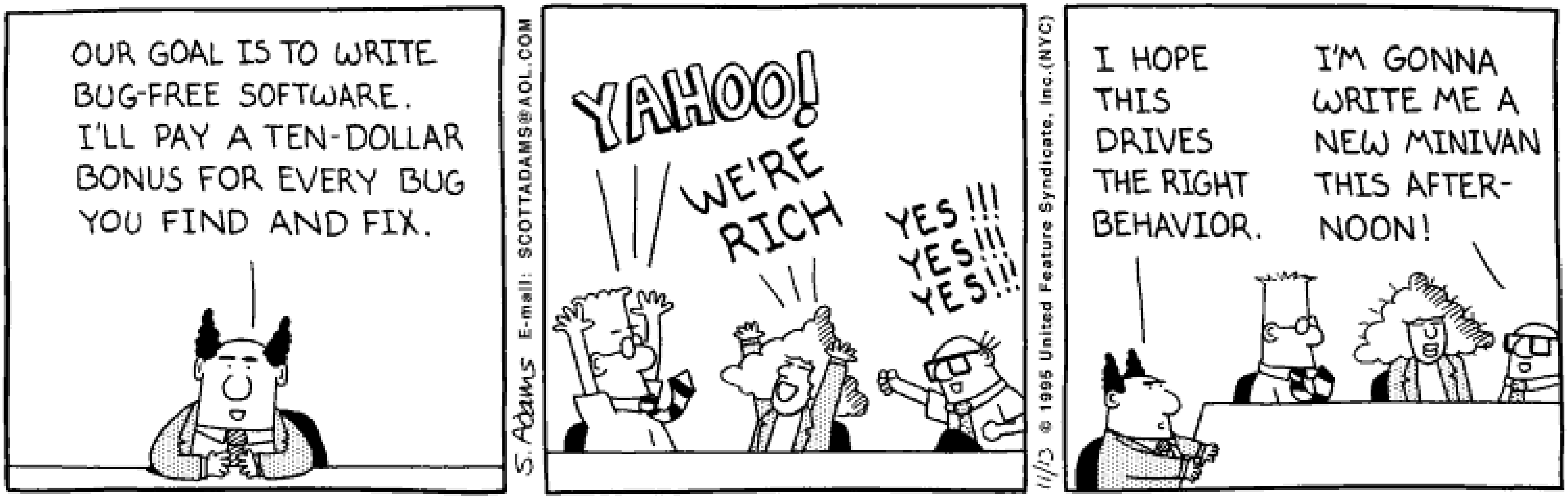
- Analysis and Design:
  - Testing begins by evaluating the OOA and OOD models
  - *How do we test OOA models (requirements and use cases)?*
  - *How do we test OOD models (class and sequence diagrams)?*

- Programming:
  - *How does OO make testing different from procedural programming?*
  - Concept of a ‘unit’ broadens due to class encapsulation
  - Integration focuses on *classes* and their context of a use case scenario

# The Bug Curve



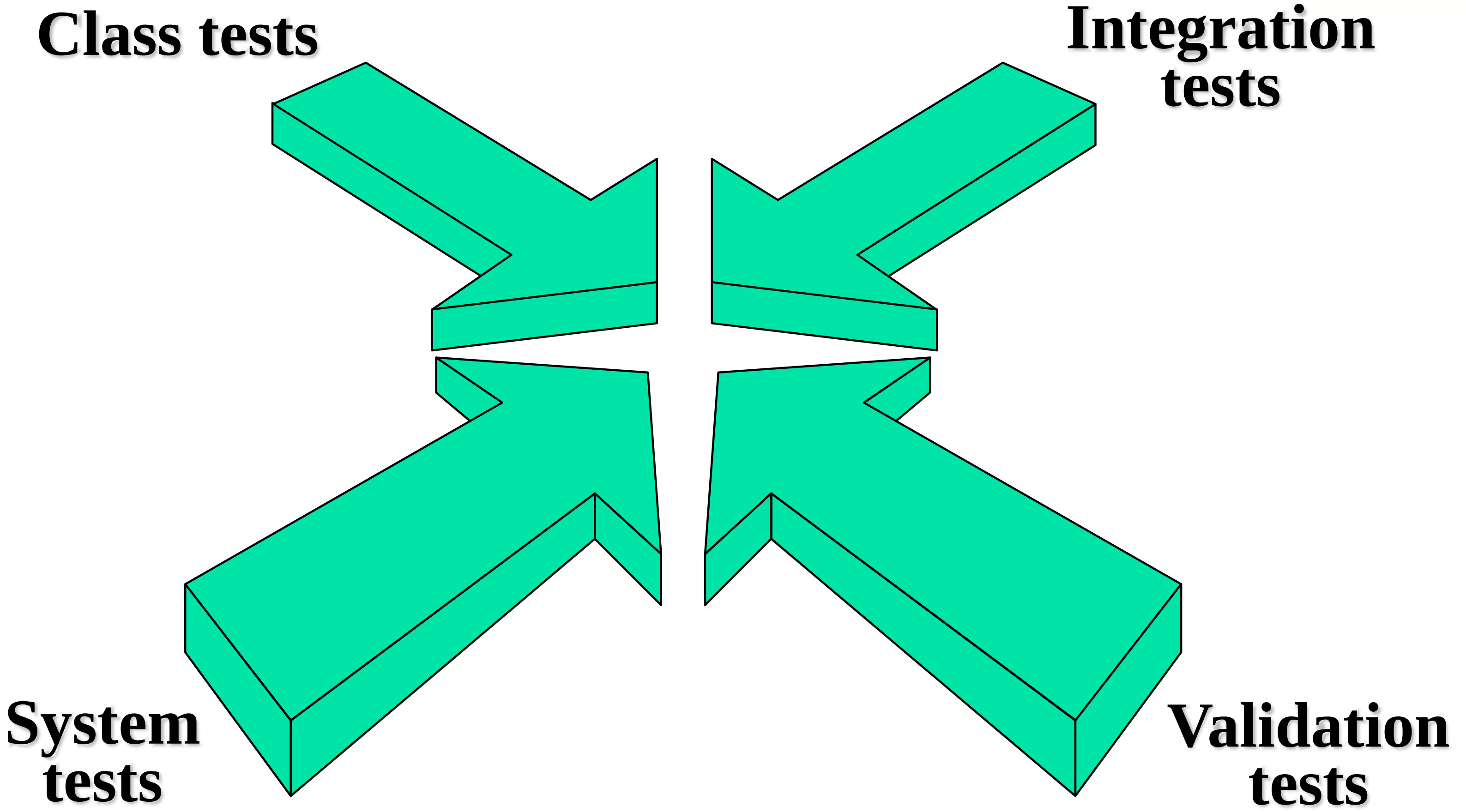
# YAHOO!


OUR GOAL IS TO WRITE BUG-FREE SOFTWARE. I'LL PAY A TEN-DOLLAR BONUS FOR EVERY BUG YOU FIND AND FIX.
S. Adams E-mail: SCOTTADAMS@AOL.COM
YAHOO!
WE'RE RICH
YES!!! YES!!! YES!!!
11/13 © 1995 United Feature Syndicate, Inc.(NYC)
I HOPE THIS DRIVES THE RIGHT BEHAVIOR.
I'M GONNA WRITE ME A NEW MINIVAN THIS AFTER-NOON!

# Testing Analysis and Design

- Syntactic correctness:
  - Are UML and ADT notation used correctly?
- Semantic correctness:
  - Does the model reflect the real world problem?
  - Is UML used as intended by its designers?
- Testing for consistency:
  - An inconsistent model has representations in one part that are not reflected in other portions of the model

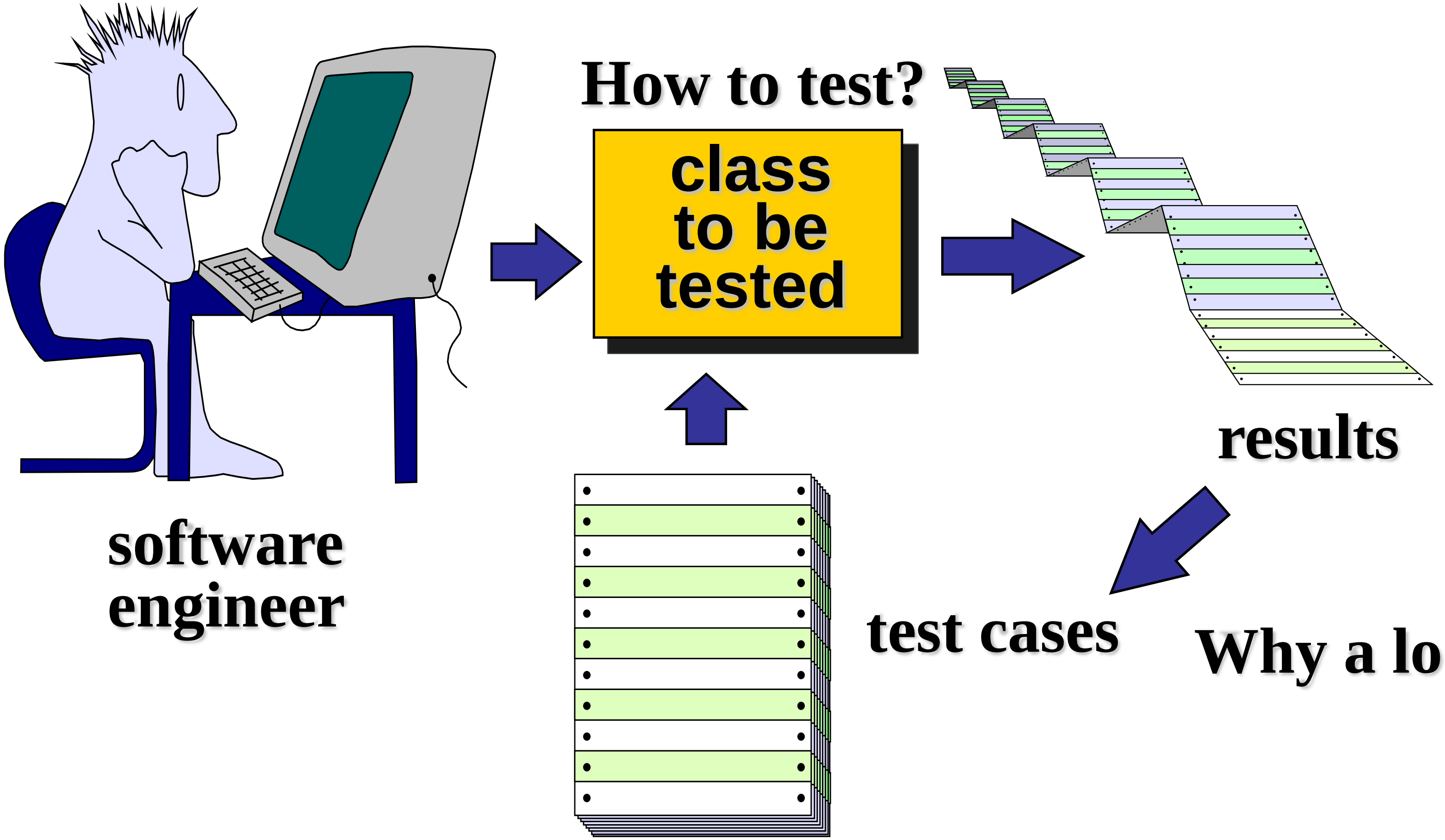
# Testing OO Code



# [1] Class (Unit) Testing

- Smallest testable unit is the encapsulated class
- Test each operation as part of a class hierarchy because its class hierarchy defines its context of use
- Approach:
  - Test each method (and constructor) within a class
  - Test the state behavior (attributes) of the class between methods
- *How is class testing different from conventional testing?*
- Conventional testing focuses on input-process-output, whereas class testing focuses on each method, then designing sequences of methods to exercise states of a class
- But white-box testing can still be applied

# Class Testing Process



# Class Test Case Design

1. Identify each test case uniquely
   - Associate test case explicitly with the class and/or method to be tested
2. State the purpose of the test
3. Each test case should contain:
   1. A list of messages and operations that will be exercised as a consequence of the test
   2. A list of exceptions that may occur as the object is tested
   3. A list of external conditions for setup (i.e., changes in the environment external to the software that must exist in order to properly conduct the test)
   4. Supplementary information that will aid in understanding or implementing the test

– Automated unit testing tools facilitate these requirements

# Challenges of Class Testing

- Encapsulation:
  - Difficult to obtain a snapshot of a class without building extra methods which display the classes’ state
- Inheritance and polymorphism:
  - Each new context of use (subclass) requires re-testing because a method may be implemented differently (polymorphism).
  - Other unaltered methods within the subclass may use the redefined method and need to be tested
- White box tests:
  - Basis path, condition, data flow and loop tests can all apply to individual methods, but don’t test interactions between methods

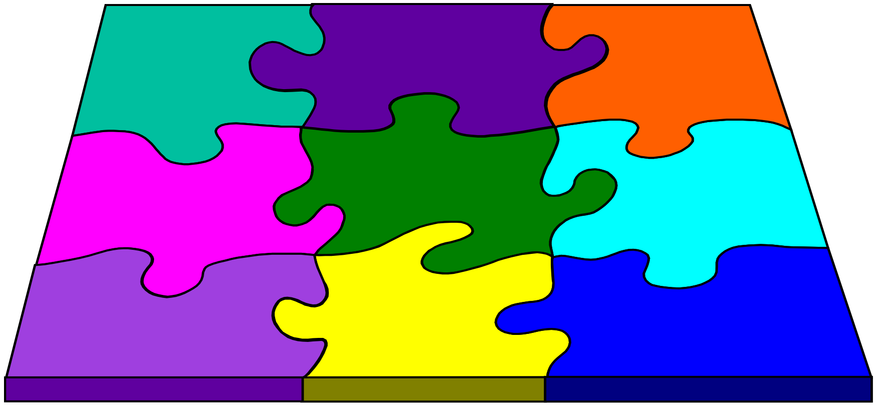
# Random Class Testing

I. Identify methods applicable to a class
II. Define constraints on their use – e.g. the class must always be initialized first
III. Identify a minimum test sequence – an operation sequence that defines the minimum life history of the class
IV. Generate a variety of random (but valid) test sequences – this exercises more complex class instance life histories

- Example:
    - An account class in a banking application has *open*, *setup*, *deposit*, *withdraw*, *balance*, *summarize* and *close* methods
    - The account must be opened first and closed on completion
    - *Open – setup – deposit – withdraw – close*
    - *Open – setup – deposit –* [deposit | withdraw | balance | summarize] – withdraw – close*. Generate random test sequences using this template

# Integration Testing

- OO does not have a hierarchical control structure so conventional top-down and bottom-up integration tests have little meaning
- Integration applied three different incremental strategies:
  - Thread-based testing: integrates classes required to respond to one input or event
  - Use-based testing: integrates classes required by one use case
  - Cluster testing: integrates classes required to demonstrate one collaboration



# Random Integration Testing

- **Multiple Class Random Testing**
  - For each client class, use the list of class methods to generate a series of random test sequences.
    Methods will send messages to other server classes.
  - For each message that is generated, determine the collaborating class and the corresponding method in the server object.
  - For each method in the server object (that has been invoked by messages sent from the client object), determine the messages that it transmits
  - For each of the messages, determine the next level of methods that are invoked and incorporate these into the test sequence

# Validation Testing

- Are we building the right product?

- Apply:
  - Use-case scenarios from the software requirements specification
  - Acceptance tests through alpha (at developer's site) and beta (at customer's site) testing with actual customers

# System Testing

**Types of System Testing:**

- Recovery testing: how well and quickly does the system recover from faults
- Security testing: verify that protection mechanisms built into the system will protect from unauthorized access (hackers, disgruntled employees, fraudsters)
- Stress testing: place abnormal load on the system
- Performance testing: investigate the run-time performance within the context of an integrated system

# Testing Tools

| **Programming Language** | **Automated testing Tool for Unit Testing** |
|---|---|
| C++ | Microsoft Unit Testing Framework for C++ |
| Java | Junit |
| .Net Programming Language | Nunit,XUnit |
| Prolog | PIUnit |
| Python | Unittest |
|  |  |

# Sample Unit Test Code in Microsoft Unit Testing Framework

```
#include "stdafx.h"
#include <CppUnitTest.h>
 #include "..\MyProjectUnderTest\MyCodeUnderTest.h"
using namespace Microsoft::VisualStudio::CppUnitTestFramework;
 TEST_CLASS(TestClassName)
{
      public: TEST_METHOD(TestMethodName)
{         // Run a function under test here.
      Assert::AreEqual(expectedValue, actualValue, L"message",
LINE_INFO());
}
}
```

# Testing Summary

***Testing can contribute to improved quality by helping the programmers to identify problems early in the development process.***