**UnitTesting**

**Names:**Cayambe Chiu, Vanessa Katherine

Chasi Lopez, Geraldi Mayte

Chiliquinga Lema, Bryan David

Coba Flores, Victor Steve

Contreras Farinango, Ian Marcelo

Criollo Gualotuña, Kevin Orlando

Cuaspa Cuaspud, Eliana Aidee

Galarza Tufiño, Andres Maximiliano

**Nrc:** 3730

**Group:** #2

**RESUME**

**Abstract.-**

Planning for Successful

Quality Control → User -Acceptance, System Test Plan, Integrate Test Plan, Unit Test Plan

Test plan

• Test plans contents (continued)

Who will do the work?

How will the testing work be managed?

How will testing be coordinated with other development activities?

Types of testing

We can categorize testing activities in three days:

By scope:

The entire system, a subsystem, a module, a single function, etc

By technique

Path test,input-validation testing,logic-based testing,stress testing, load testing,etc

By source of test data

Source code, designs, specifications,requirements, customers,random, ect.

Types of testing : by scope

**Testing in the Small:** Involves testing code elements or coherent work units, such as procedures, functions, an abstract data types, classes, and small modules

**Testing in the Large:** involves testing a set of work units that have been assembled to form a more complex component, subsystem, or a system.

Types of testing : by technique

* Reduce the number of tests
* Each testing technique focuses on a different aspect of the software
* Different techniques uncover different faults
* Regardless of technique, test cases consist of

Setup (or stimulus)

Predicated results

Comparison of observed results to predicated results

Testing Techniques

The different techniques focus on different aspects of the system, there are differences in

how test cases are set up

how predicated results are documented

how the actually results are compared to the predicted results

* Some traditional techniques

Path testing / all are execute l

Input-validation and syntax testing / right format

Transaction-flow testing / all or nothing

Logic-based testing / Boolean algebra

State-transition testing / finite state machines

Stress testing / beyond normal capacity

Load-balance testing / multiple servers or services

Types of testing : source of test data

• Requirements definition

• Requirements specifications

• Architectural designs

• Detailed designs

• Code • Extreme or abnormal data

• Random, pseudo-random, or generated data

• Actual or existing data

• Extrapolations from prior test cases

Source of test data

* White-box testing

the designs or code are used as a source of information

* Black-box testing

the designs and code are not consulted when designing and writing test cases

Selection of Appropriate Test Data

* Fundamental questions:

What are the possible sources of failure in a program?

What test data should be selected to demonstrate that failures do not arise from these sources?

Alternate question: What and how many test cases are good enough?

Test Data selection

Definitions

Let F be a program

Let D be the domain of possible inputs to that program and d be an instance of D

Let F(d) be the results of executing F with input d

Let OUT be a relation, and more precisely a predicate, between D and the range of F(d), such that OUT(d, F(d)) is true if and only if F(d) is the expected result for executing F with input d

Ideal test (set of test cases) T

one that succeeds if and only if there are no errors in the program

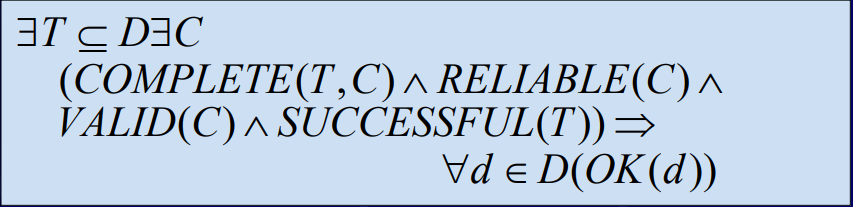
Exhaustive testing: T = D

• Thorough testing: T satisfies some set of test-selection criteria, C l COMPLETE(T, C)

• The trick is now to choose C so the testing results are consistent and meaningful RELIABLE(C)

VALID(C)

Fundamental Theorem of testing



The theorem says that tests satisfying COMPLETE(T, C) where C is reliable and valid are “thorough” in an appropriate sense.

Issues Regarding Test-data

Selection Criteria

Choosing C

The criteria depends on the technique

Therefore, choose techniques first and then choose selection criteria for each technique

• Proving RELIABLE(C)

• Proving VALID(C)

• To get a handle on these three issues, we must understand the nature and causes of software errors

Test-Case Selection Criteria for Common Techniques

Criteria for Path Testing

A set of test cases, T, executes all lines of code

T tries all branches of conditions

T tries the boundary cases of loops Criteria for Input-Validation Testing

For each input domain (all valid and invalid possibilities),

T tries at least one value from each subset in the partitioning of that domain   
 T tries bound conditions

Putting Testing into Practice

Terminology

Failures - observed errors or inconsistencies

Faults - errors in the software, documentation, or other work product

Faults are commonly referred to as bugs Testing focuses on finding failures

Debugging focuses on finding faults

Thorough testing must expose failures arising from performance or logical faults

Types of Fault

• Performance Errors: software doesn’t produce results within specified time and space limitations

• Logic Errors: software produces incorrect results independent of the time and space required

Sources of Errors

• Errors can occur in any phase or activity of the development process

• Many of these errors may manifest themselves during implementation

In general, the earlier you can detect a fault, the cheaper and easier it is to fix

For discussion purposes, we will refer to any fault that causes a failure in the software system as a software error, regardless of the fault’s source.

• We will focus on testing techniques that uncover software errors.

Sources of Software Errors

• Three sources of software errors mentioned by Goodenough and Gerhart

Missing Control Flow Paths

Inappropriate Path Selection

Inappropriate or Missing Action

• Others?

What are some of the typical mistakes that you make?

What are some of the hardest to find?

Do the types of mistakes that you make depend on the development environment?

• Memory management errors

Confusing references/pointers

Out-of-bounds references

Initialization

• Understanding semantics of other components

• Configuration and environment

• Timing errors

Unit testing

We must test every unit /component of a software system

We must validate each unit A Unit is the smallest testable part of a software, usually, a method (function)

Inputs are provided, results are expected add(a,b)

Test cases

“It is a specification of the inputs, execution conditions, testing procedure, and expected results that define a single test to achieve a particular software testing objective” Wikipedia

At least two cases: one positive test and one negative test

Expected output must be defined in advance

We may need as many test cases as needed, but not too many

**QUICK START TO UNIT TESTING**

**PROBLEM: HOW CAN “I” KNOW**

**MY CODE IS CORRECT**

Consider the following class:

Validation:

Is it doing the right thing?

Verification:

Is it doing that thing

right?

**SAMPLE TEST CASES**

SideLengths = …

1.{ 3.0, 4.0, 5.0 }

2.{ 3.432, 4.525, 5.236 }

3.{ 2.0, 3.0 }

1.{ 1.0, 1.0, 1.0, 1.0 }

2.{-1.0, 2.0, 3.0 }

3.{10, 2, 2}

1.{2, 2, 2}

2.{4 , 4, 3}

3.{null, 3, 2}

1.{2, null, 3}

2.{A, 2, 3}

3.{}

1.null

**2.?? More ??**

**CONDUCTING A TEST**

**Three general steps:**

1.Setup:Ensure the system is in a known state,state, e.g.,

create a triangle object with specific sides

2.Stimulate: Execute the thing you are testing, e.g., the

ComputeArea() method

3.Observe:Compare the actual results with the expected results.

**MAKING A TEST EXECUTABLE**

All three steps for specific test cases can be captured as an executable

method or function.

**MAKING A TEST EXECUTABLE**

Encapsulation of comparisons into Assertions

**USING TESTING FRAMEWORKS**

• A testing framework provides tools for

• Setting up test cases

• Making observations, e.g., the Assert class

• Organizing test cases

• Executing test cases

• Examples:

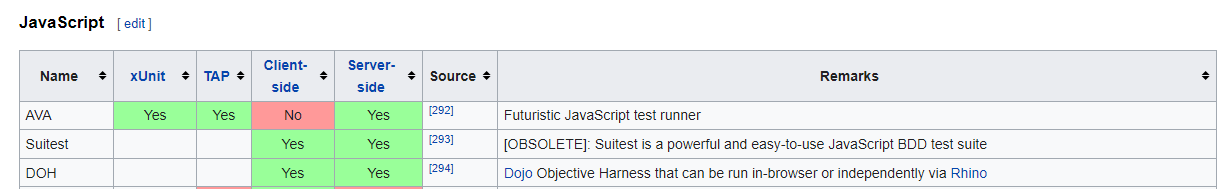
• Java: JUnit

• C# (or more generally, anything .Net) : VS Unit Testing Framework, NUnit

# **Unit testing frameworks**

**COLUMNS(CLASSIFICATION)**

* **Name**: This column contains the name of the framework and will usually link to it.
* **xUnit**: This column indicates whether a framework should be considered of [xUnit](https://en.wikipedia.org/wiki/XUnit) type.
* **TAP**: This column indicates whether a framework can emit [TAP](https://en.wikipedia.org/wiki/Test_Anything_Protocol) output for TAP-compliant [testing harnesses](https://en.wikipedia.org/wiki/Test_harness).
* **SubUnit**: This column indicates whether a framework can emit [SubUnit](https://en.wikipedia.org/wiki/Subunit_(format)) output.
* **Generators**: Indicates whether a framework supports [data generators](https://en.wikipedia.org/wiki/Test_bench).
* **Fixtures**: Indicates whether a framework supports test-local fixtures. Test-local fixtures ensure a specified environment for a single test.
* **Group fixtures**: Indicates whether a framework supports group fixtures.
* **MPI**: Indicates whether a framework supports message passing via MPI - commonly used for high-performance scientific computing.
* **Other columns**: These columns indicate whether a specific language / tool feature is available / used by a framework.
* **Remarks**: Any remarks.



**Test Case Selection**

Selection of Approriate Test

Data

Fundamental questions:

-What are the possible sources of failure in a program ?

-What test data should be selected to demonstrate that failures do not arise from these sources?

-Alternate questions:What and how many test cases are good enough ?

**Test Data Selection**

Definitions

-Let F be a program,subsystem,class,method,procedure,or some other work artifact

-let D be the domain of possible inputs to that program and d be an instace of D

-Let F(d) be the results of executing F with inpunt d

-Let OUT be a predicate between D and F(d),such that OUT (d,F(d)) is true if and only if F(d)is the expected results for executing F with input d

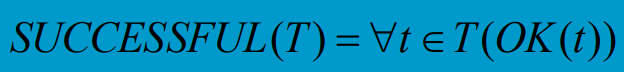
-Let T be set of test cases,where T ⊆ D , and is called a test site

**Convenient Notations**

Shorthand for success/failure of a test case, when F is understood:

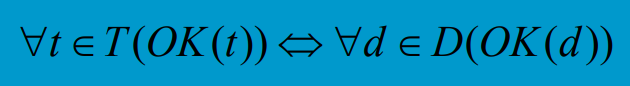


Shorthand for success/failure of a whole test suite, T:



**Test Data Selection**

Ideal test suite, T, is a set of test cases such that it succeeds if and only if there are no errors in the program



**Test-Data Selection**

Exhaustive testing: T = D

• Thorough testing: T satisfies some set of test-selection criteria, C

– COMPLETE(T, C)

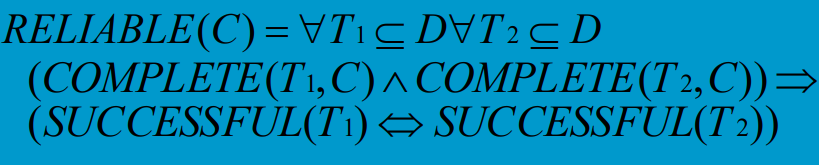
• The trick is now to choose the criteria, C, so the testing results are consistent and meaningful, and common close to being ideal

– RELIABLE(C)

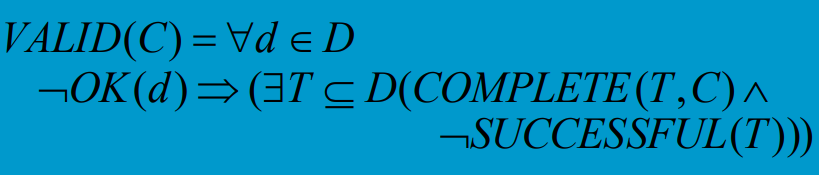
– VALID(C)

**Definitions about Criteria**

Criteria reliability:

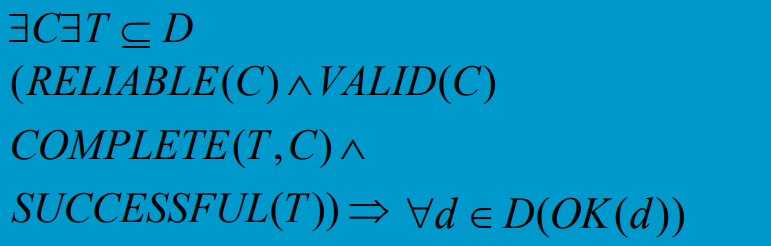


Criteria validity:



**Fundamental Theorem of Testing**

Theorem:



The theorem says that if there exists some criteria that is reliable and valid, a test suite that satisfies the criteria, and the test suite is successful, then all of D should be successful. • In this case, the test suite is “thorough”

**Issues Regarding Test-data Selection Criteria**

Choosing C

– The criteria depends on the testing technique

– Therefore, choose techniques first and then choose selection criteria for each technique

• Proving RELIABLE(C)

• Proving VALID(C)

• To get a handle on these three issues, we must understand the nature and causes of software errors

**Sample Criteria for Modified Path Testing**

Choose T such that

– Every line of code is executed

– Every possible branch of a condition tried

– Boundary conditions for loops are tried

– If concurrent execute is possible, also

• Every possibility on entry conditions of every critical section are tried

**Sample Criteria for Input Validation Testing**

• Partition D into meaning sub-domains

– Consider representative values of D, independent of any constraints or embedded language

– Determine if there are constraints on D

• If there is, consider valid and invalid values

– Determine if there is an embedded language

• If there is, consider legal and illegal value with respect to the language ! Choose T such that

– There is one test case for each sub-domain

**Nature and Cause of Software Errors**

Terminology

– Failures - observed errors or inconsistencies

– Faults - errors in the software, documentation, or other work product

What’s a “bug”?

– This is an overloaded terms that people informally use for both failures and faults

**Sources of Errors**

**•** Errors can occur in any phase or activity of the development process

• Many of these errors may manifested themselves during implementation

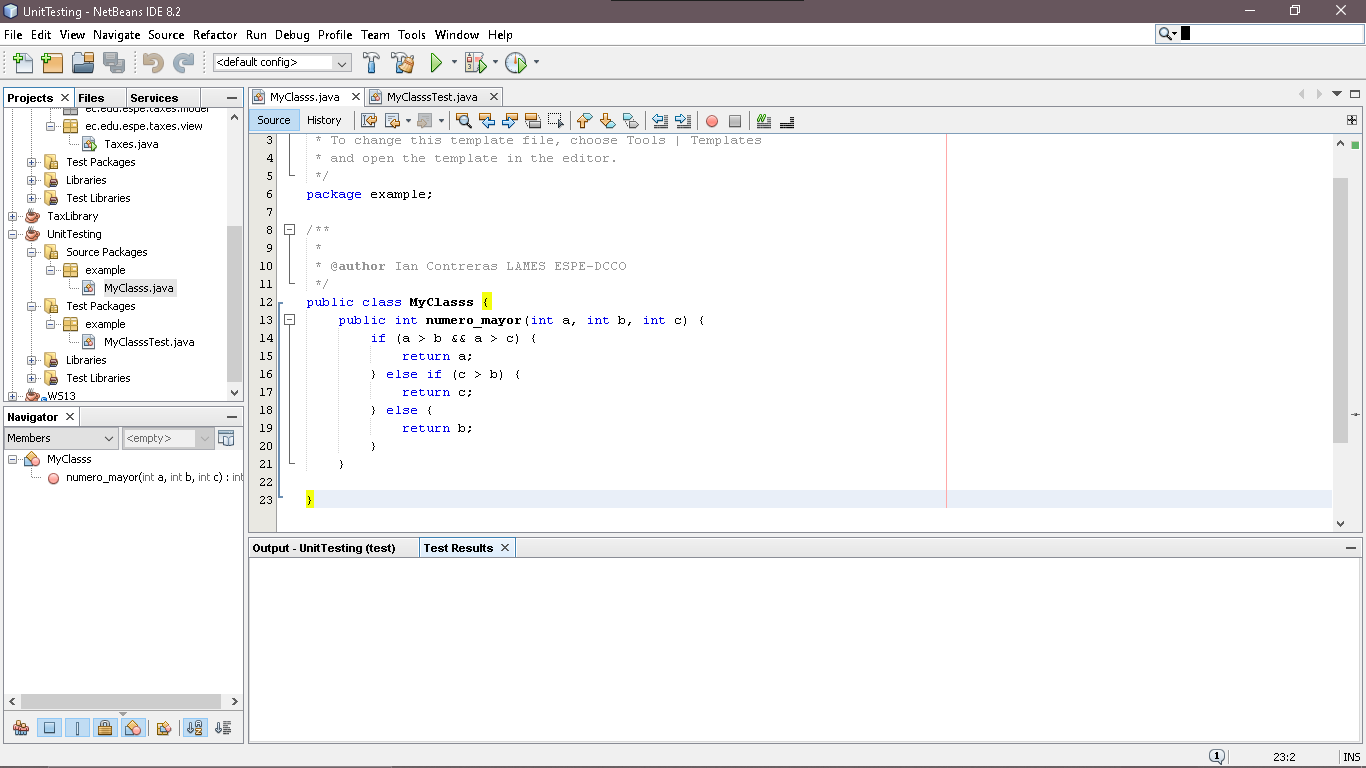
– In general, the earlier you can detect a fault, the cheaper and easier it is to fix

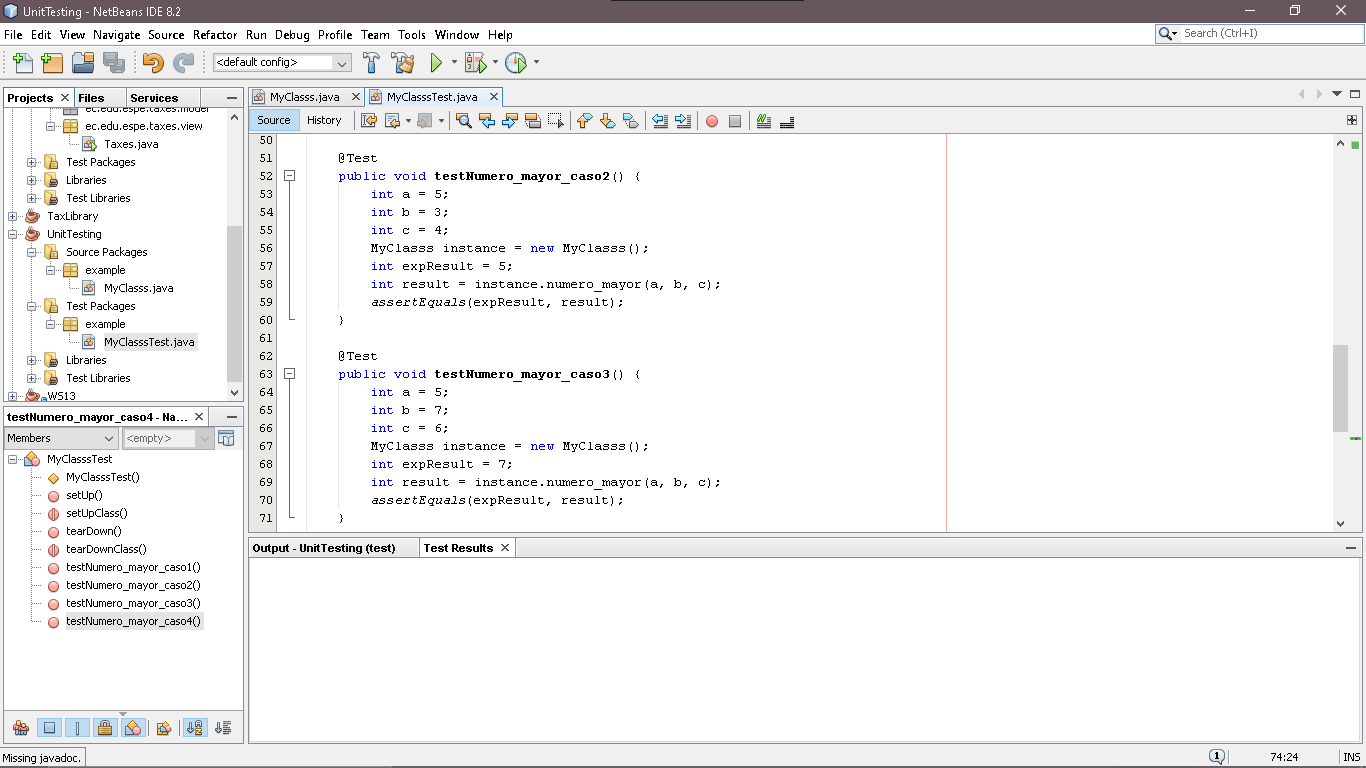
– For discussion purposes, we will refer to any fault that causes a failure in the software system as a software error, regardless of the fault’s source.

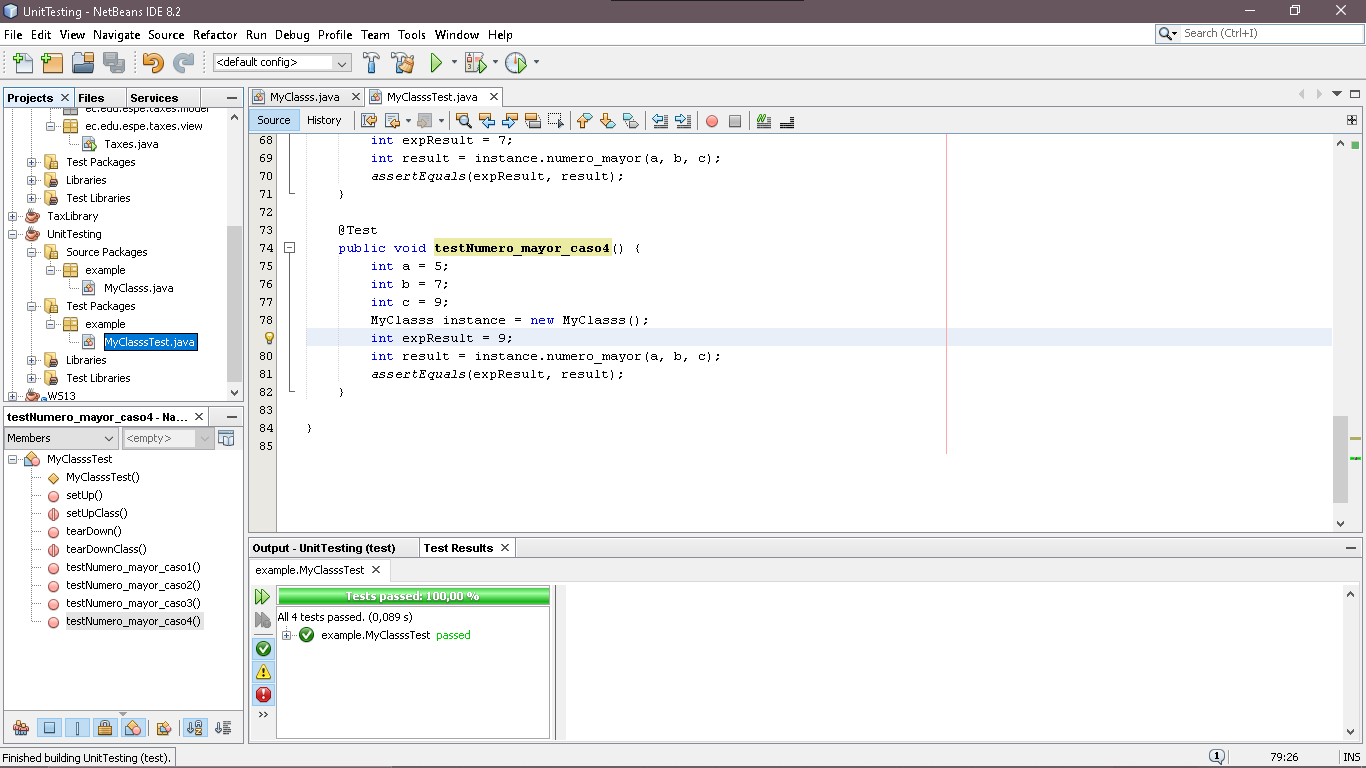
• We will focus on testing techniques that uncover software errors

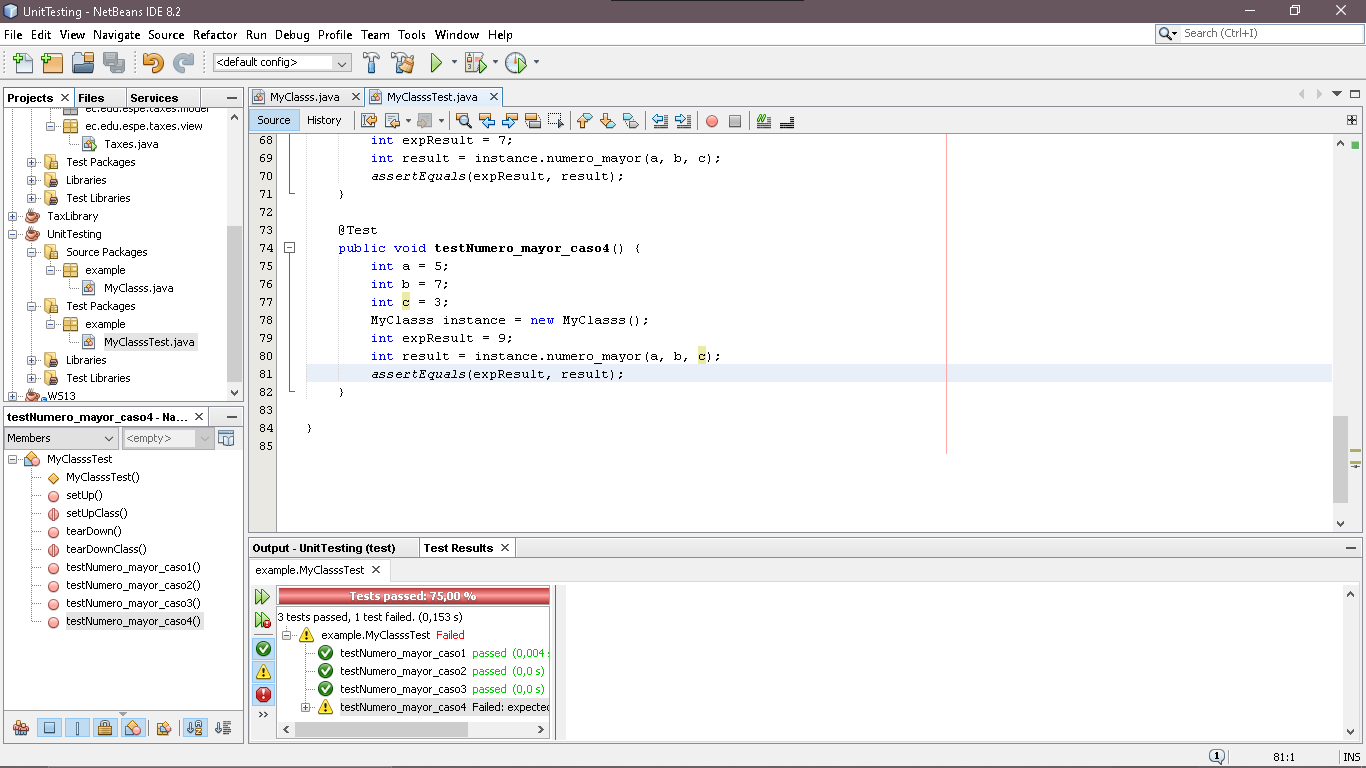
**CODE.**

**Example.-**









**Questions.-**

1. We can categorize testing activities in three ways: By scope, By technique,........
2. By source of test data.
3. By data source
4. By test data
5. The definition to what types faults corresponds:

Software doesn't produce results within specified time and space limitations.

1. Performance Errors
2. Logic Errors
3. Source of Errors

3.Complete the following statement.

The libraries that take care to do the unit testing are:….....

1. gson-2.8.6 and javacsv
2. hamcrest-core-1.3 and junit-4.11
3. junit-4.11 and javacsv
4. gson-2.8.6 and hamcrest-core-1.3

4. What is the shorthand for success/failure of a whole test suite?

1. D
2. F
3. T

5. Choose one of the sources of software errors by Goodenough and Gerhart

1. Proving Reliable(C)
2. Missing Control Flow Paths
3. Initialization