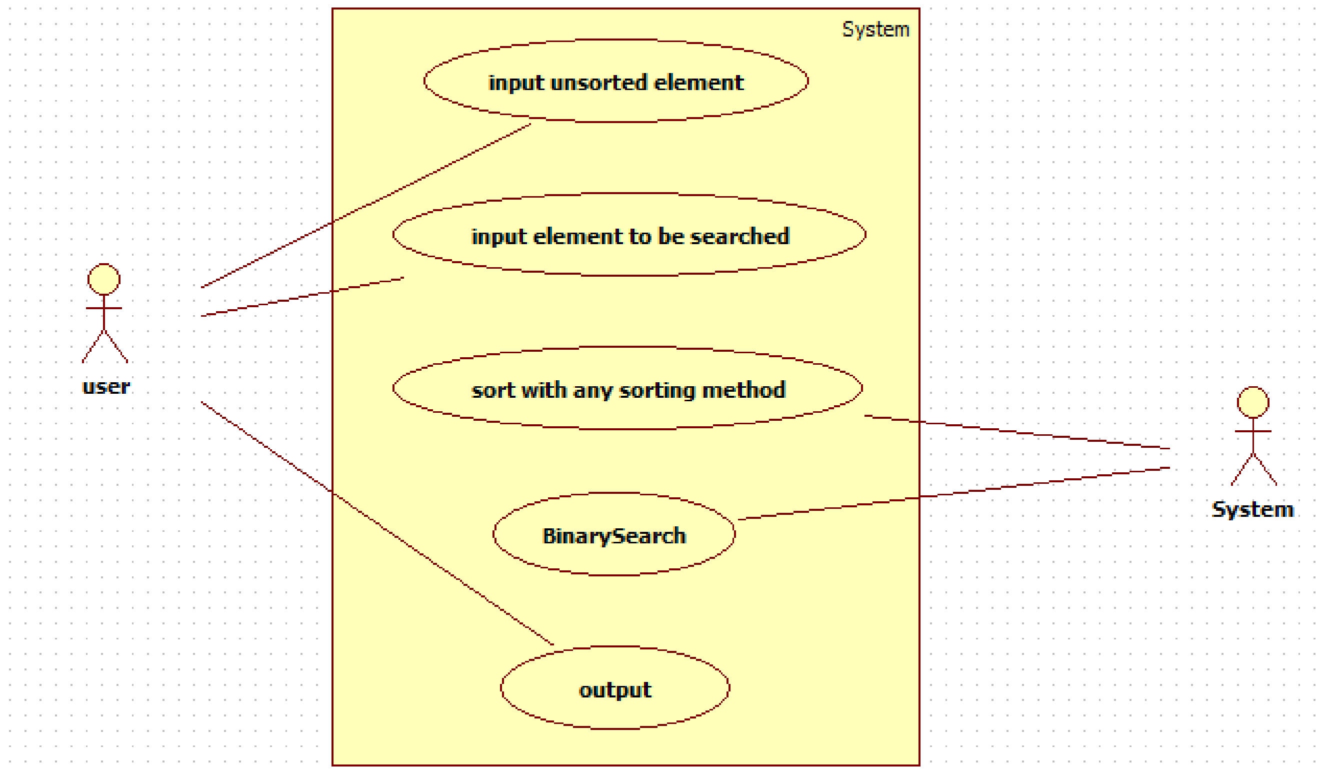
Assignment 1 Binary Search

UML USE-CASE diagram for Binary Search:-



Relative Efficiency:

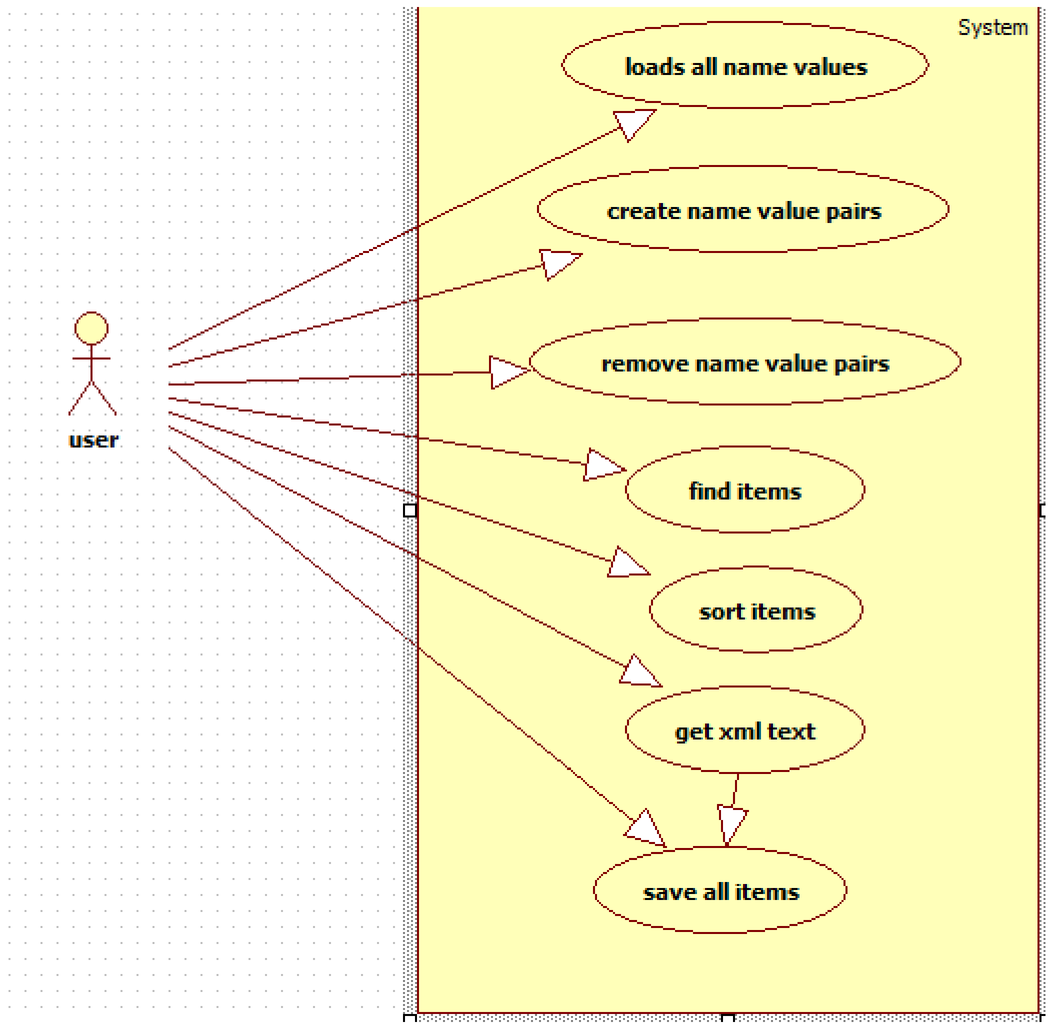
Class	**Search algorithm**
Worst case performance	O(log n)
Best case performance	O(1)
Average case performance	O(log n)
Worst case space complexity	O(1)

Patterns:-

- ✓ FAÇADE
- ✓ Command

Assignment 2 Quick Sort

UML USE-CASE diagram for Quick Sort:-



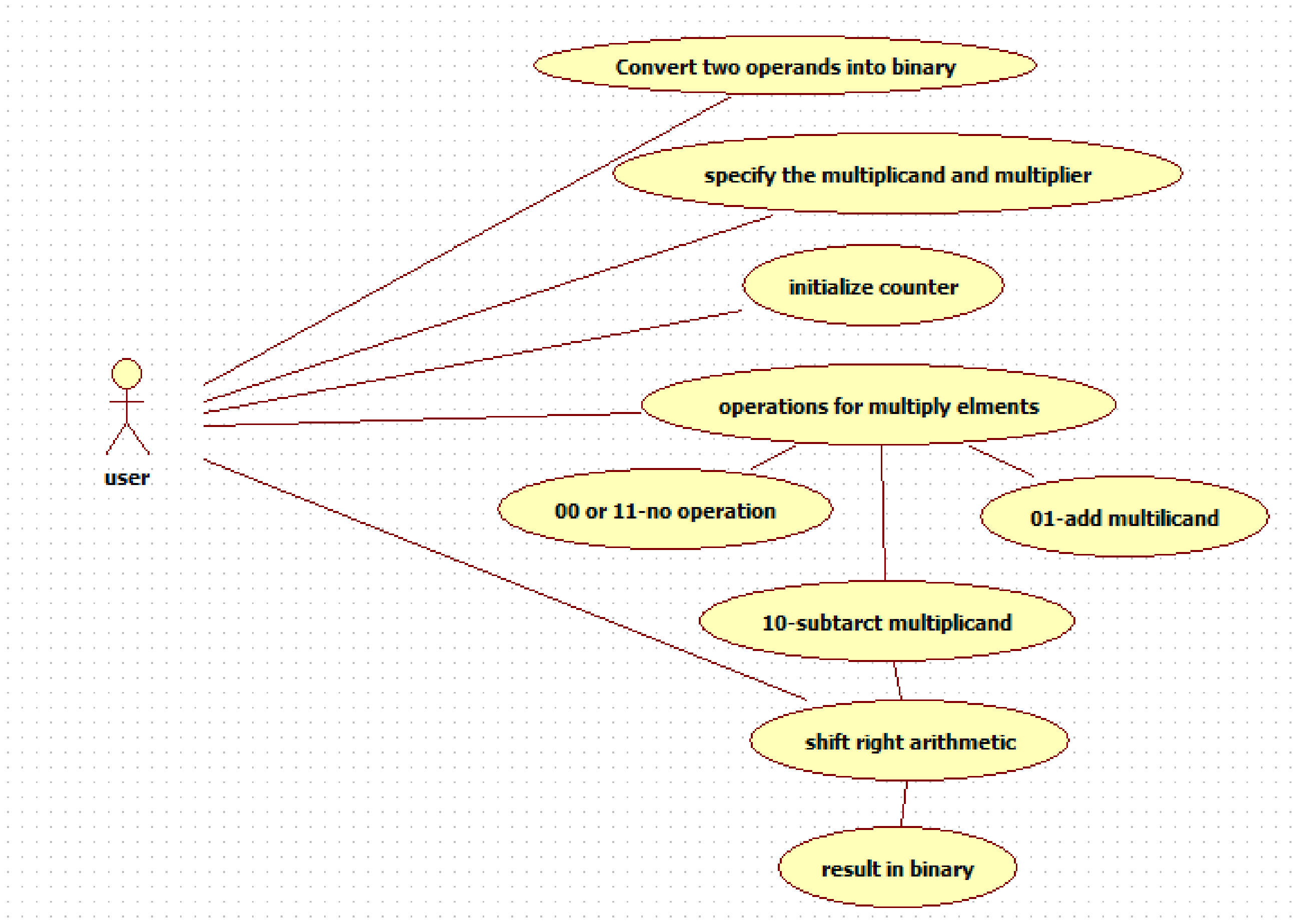
Relative efficiency:

Class	**Search algorithm**
Worst case performance	$O(N^2)$
Best case performance	O(NlogN)
Average case performance	O(NlogN)

Patterns:-

- ✓ FAÇADE
- ✓ Command

Assignment 3 Booth Multiplication



Booth's algorithm can be implemented by repeatedly adding (with ordinary unsigned binary addition) one of two predetermined values A and S to a product P, then performing a rightward arithmetic shift on P. Let m and r be the multiplicand and multiplier, respectively; and let x and y represent the number of bits in m and r.

1. Determine the values of A and S, and the initial value of P. All of these numbers should have a length equal to (x + y + 1).

1. A: Fill the most significant (leftmost) bits with the value of m. Fill the remaining (y + 1) bits with zeros.

2. S: Fill the most significant bits with the value of (-m) in two's complement notation. Fill the remaining (y + 1) bits with zeros.

3. P: Fill the most significant x bits with zeros. To the right of this, append the value of r. Fill the least significant (rightmost) bit with a zero.

2. Determine the two least significant (rightmost) bits of P. 1. If they are 01, find the value of P + A. Ignore any overflow. 2. If they are 10, find the value of P + S. Ignore any overflow. 3. If they are 00, do nothing. Use P directly in the next step. 4. If they are 11, do nothing. Use P directly in the next step.

3. Arithmetically shift the value obtained in the 2nd step by a single place to the right. Let P now equal this new value.

4. Repeat steps 2 and 3 until they have been done y times. 5. Drop the least significant (rightmost) bit from P. This is the product of m and r.

Risk Analysis:-

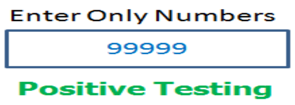
Connection refused between client and server.

Server Down

Redundant entry

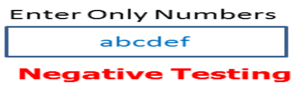
Software Testing:- **Software testing** is process of verifying and validating the software or application and checks whether it is working as expected. The intent is to find defects and improve the product quality. There are two ways to test the software viz, Positive Testing and Negative Testing.

Positive Testing:- can be performed on the system by providing the **valid data as input.** It checks whether an application behaves as expected with the positive input. . This is to test to check the application that does what it is supposed to do so.



There is a text box in an application which can accept only numbers. Entering values up to 99999 will be acceptable by the system and any other values apart from this should not be acceptable. To do positive testing, set the valid input values from 0 to 99999 and check whether the system is accepting the values.

Negative Testing:- Negative Testing can be performed on the system by providing **invalid data as input**. It checks whether an application behaves as expected with the negative input. This is to test the application that does not do anything that it is not supposed to do so. For example –



For the example above Negative testing can be performed by testing by entering alphabets characters from A to Z or from a to z. Either system text box should not accept the values or else it should throw an error message for these invalid data inputs.

In both the testing, following needs to be considered:

- Input data
- Action which needs to be performed
- Output Result

In this Assignment we perform both Positive and Negative Testing so we write the test cases for same.

Let's also get some ground rules to make sure we design good positive and negative scenarios.

Requirements:

- The Multiplicand has to be a number and its mandatory parameter.
- The Multiplier has to be a number and its mandatory parameter.

- The input number can have only 0-9 digit only. No alphabet special characters are allowed.
- Input number can be only two digit.

Positive test scenarios: Below are some positive testing scenarios for this particular pane.

Test Case ID	M	R	Expected Output(P)	Test Case Pass/Fail	Description
1	-14	a	Error	Fail	Entering text instead of numbers
2	0	252	0	Fail	Entering out of boundary values
3	-141	555	Wrong Answer	Fail	Entering out of range values

Negative test scenarios: Below are some negative testing scenarios for this particular pane.

Test Case ID	M	R	Expected Output(P)	Actual Output	Test Case Pass/Fail
1	-14	5	70	70	Pass
2	5	-14	70	70	Pass
3	-14	-5	-70	-70	Pass
4	14	5	70	70	Pass

Relative Efficiency:

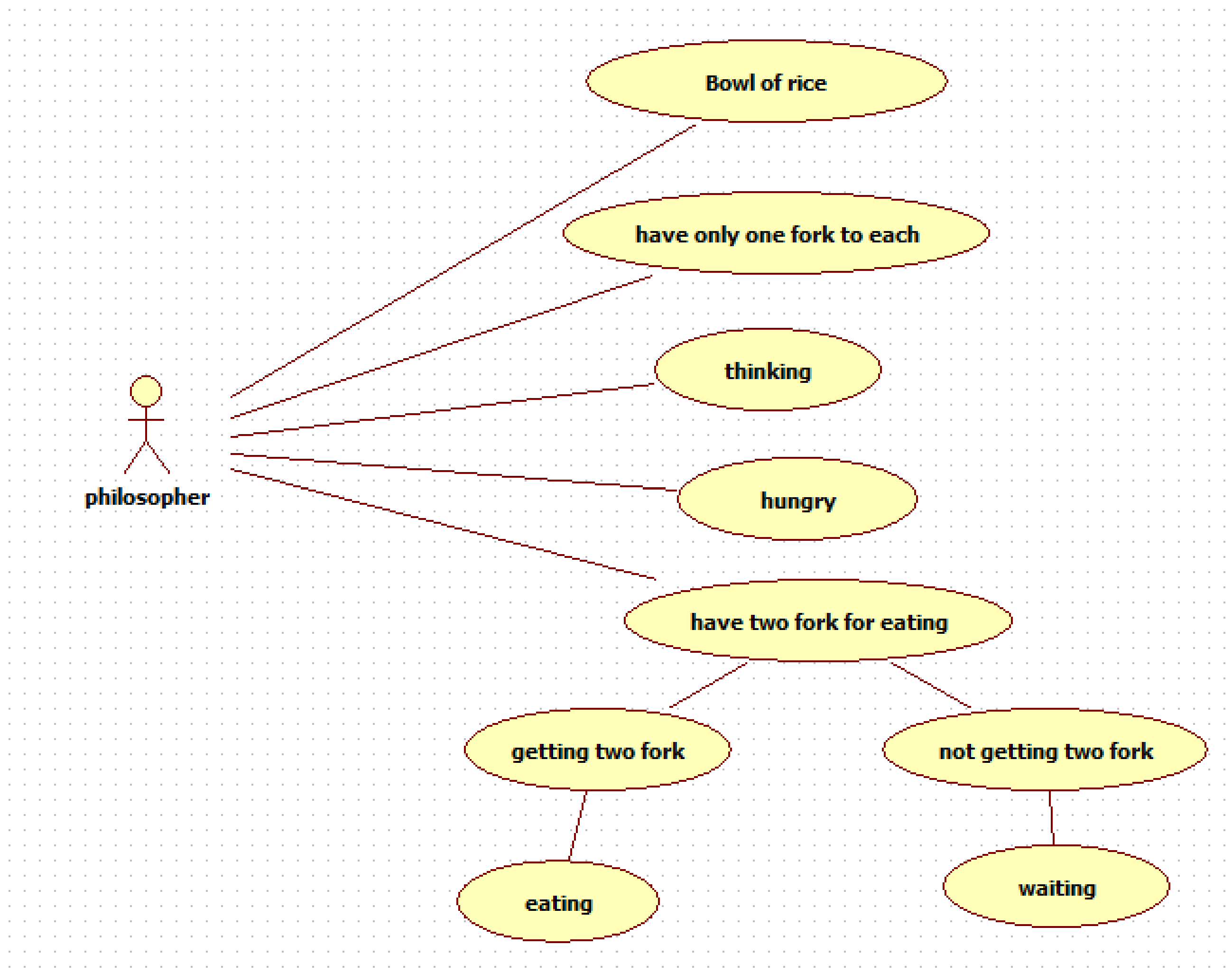
Class	**Search algorithm**
Worst case performance	O(log n)
Best case performance	O(1)
Average case performance	O(log n)
Worst case space complexity	O(1)

Design Patterns

In this assignment Socket used to establish the communication between client and server.

1. FAÇADE pattern. 2. Command Pattern.

Assignment 4 Dinning Philosopher Problem



Risks (Architectural):

- If synchronization will not be there then database will be in deadlock.
- If database is lost then there will be a problem.
- Sensor machine down.
- Sensors not active.

Design Patterns:

1. FAÇADE pattern.

Reliability Testing:-

Test-retest reliability is the most common measure of reliability. In order to measure the test-retest reliability, we have to give the same test to the same test respondents on two separate occasions. We can refer to the first time the test is given as T1 and the second time that the test is given as T2. The scores on the two occasions are then correlated. This correlation is known as the **test-retest-reliability coefficient**, or the **coefficient of stability**.

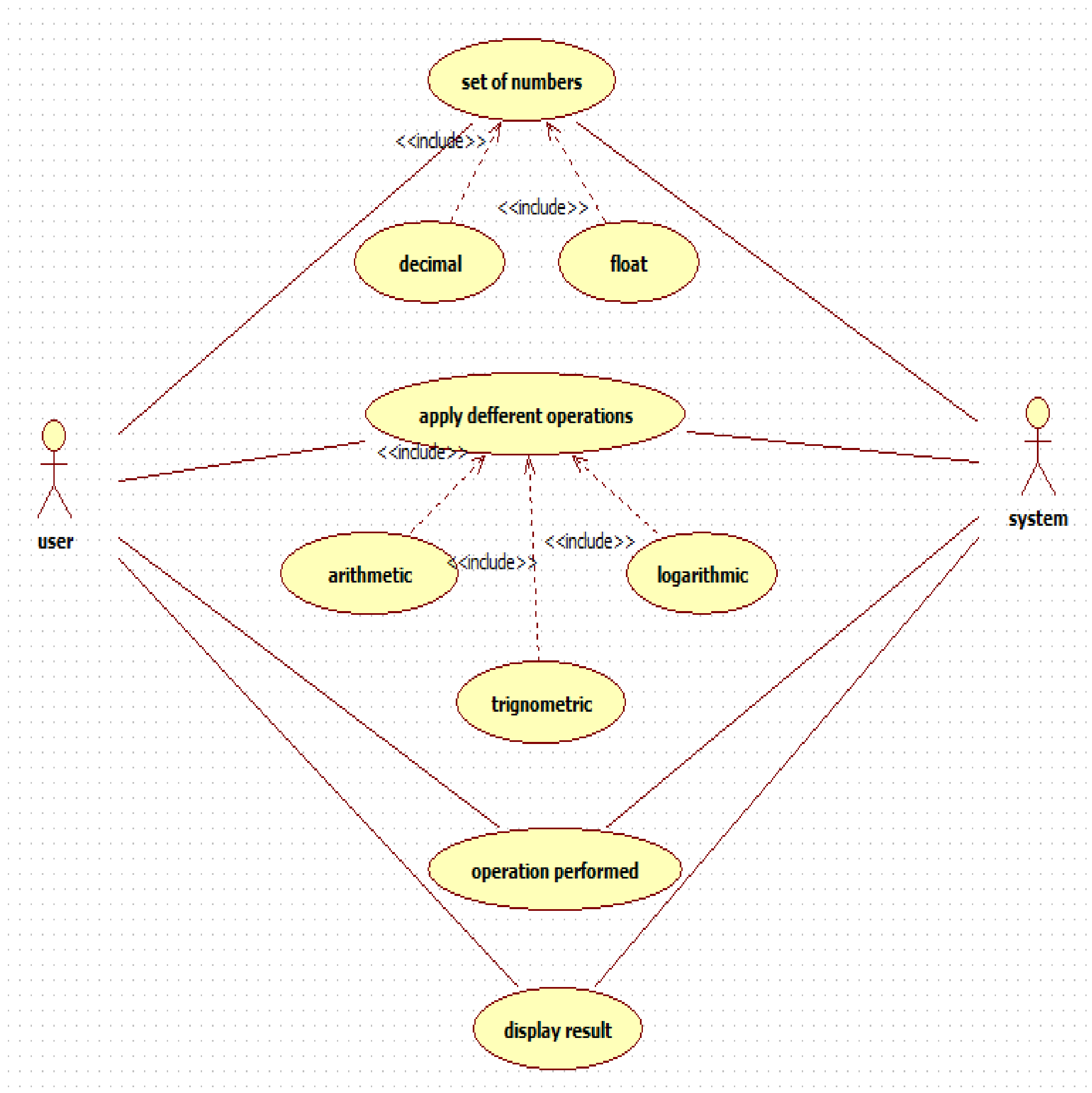
The closer each respondent's scores are on T1 and T2, the more reliable the test measure (and the higher the coefficient of stability will be). A coefficient of stability of 1 indicates that each respondent's scores are perfectly correlated. That is, each respondent score the exact same thing on T1 as they did on T2. A coefficient correlation of 0 indicates that the respondents' scores at T1 were completely unrelated to their scores at T2; therefore the test is not reliable.

So, how do we interpret the coefficients of stability that are between 1 and 0? The following guidelines can be used:

- 0.9 and greater: excellent reliability
- Between 0.9 and 0.8: good reliability
- Between 0.8 and 0.7: acceptable reliability
- Between 0.7 and 0.6: questionable reliability
- Between 0.6 and 0.5: poor reliability
- Less than 0.5: unacceptable reliability

Test ID	**Test case**	**Expected Output**	**Actual Output**	**Test Case**
1	Check the mutual exclusion in case the number of sensors are 10	No deadlock. All sensors write to the DB.	No deadlock. All sensors write to the DB.	Pass
2	Check the mutual exclusion in case the number of sensors are 20	No deadlock. All sensors write to the DB.	No deadlock. All sensors write to the DB.	Pass
3	Execute TC 1 again	No deadlock. All sensors write to the DB.	No deadlock. All sensors write to the DB.	Pass

Assignment 5 Scientific Calculator



Design Patterns:

1. Observer pattern.
2. Strategy pattern.

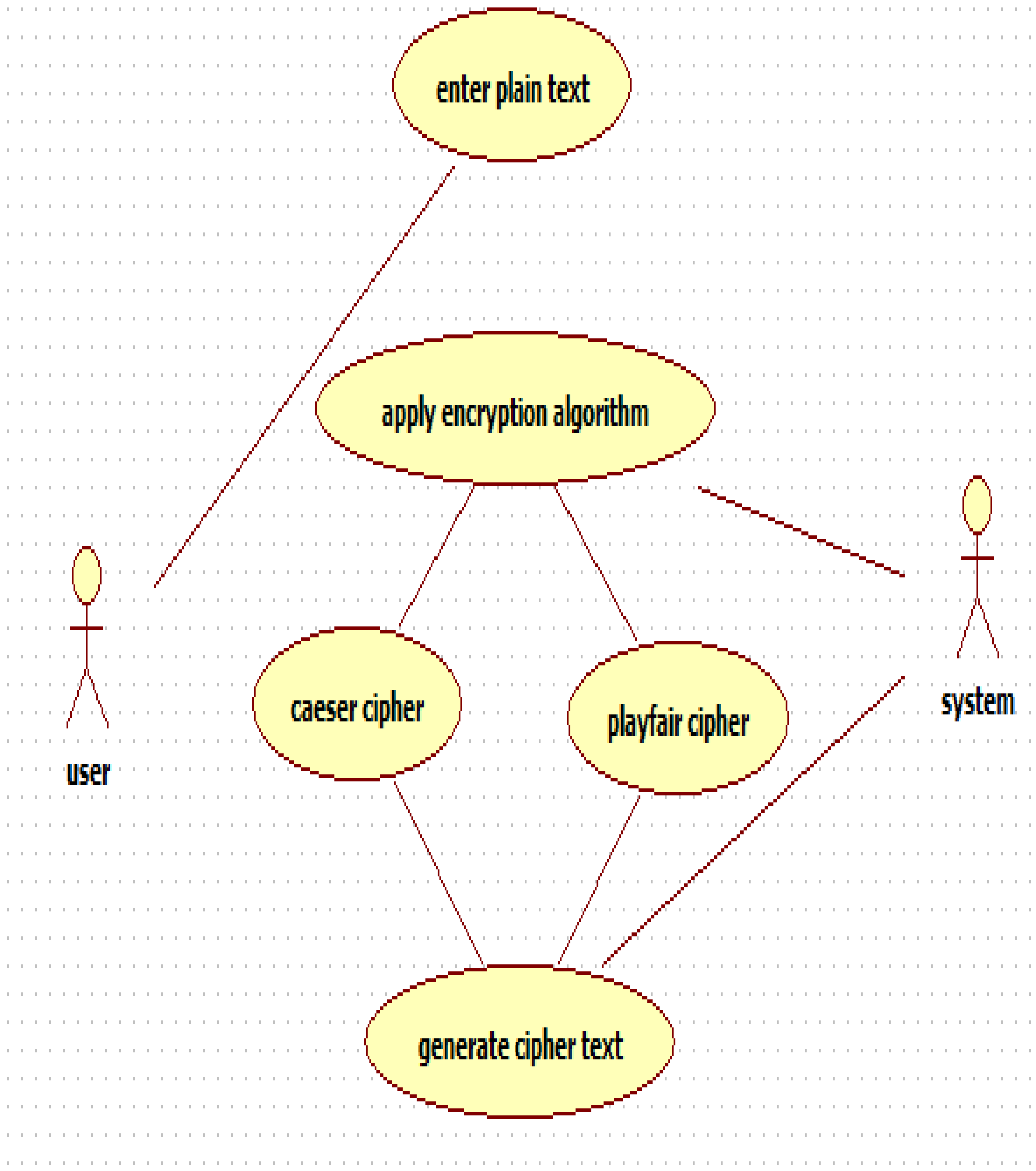
Test Cases: Positive Test cases

Test ID	**Test case Description**	**Actual Value**	**Expected Value**	**Result**
1	Check the addition of two numbers	Addition Result	Addition result	True
2	Any number multiply by Zero.	zero	zero	True
3	Multiplication of two negative numbers	Result must be positive	Result must be positive	True
4	Sin 0	0	0	True
5	Sin 90	1	1	True
6	Cos 0	1	1	True
7	Cos 90	0	0	True
8	Sin 30	0.5	0.5	True
9	Cos 60	0.5	0.5	True

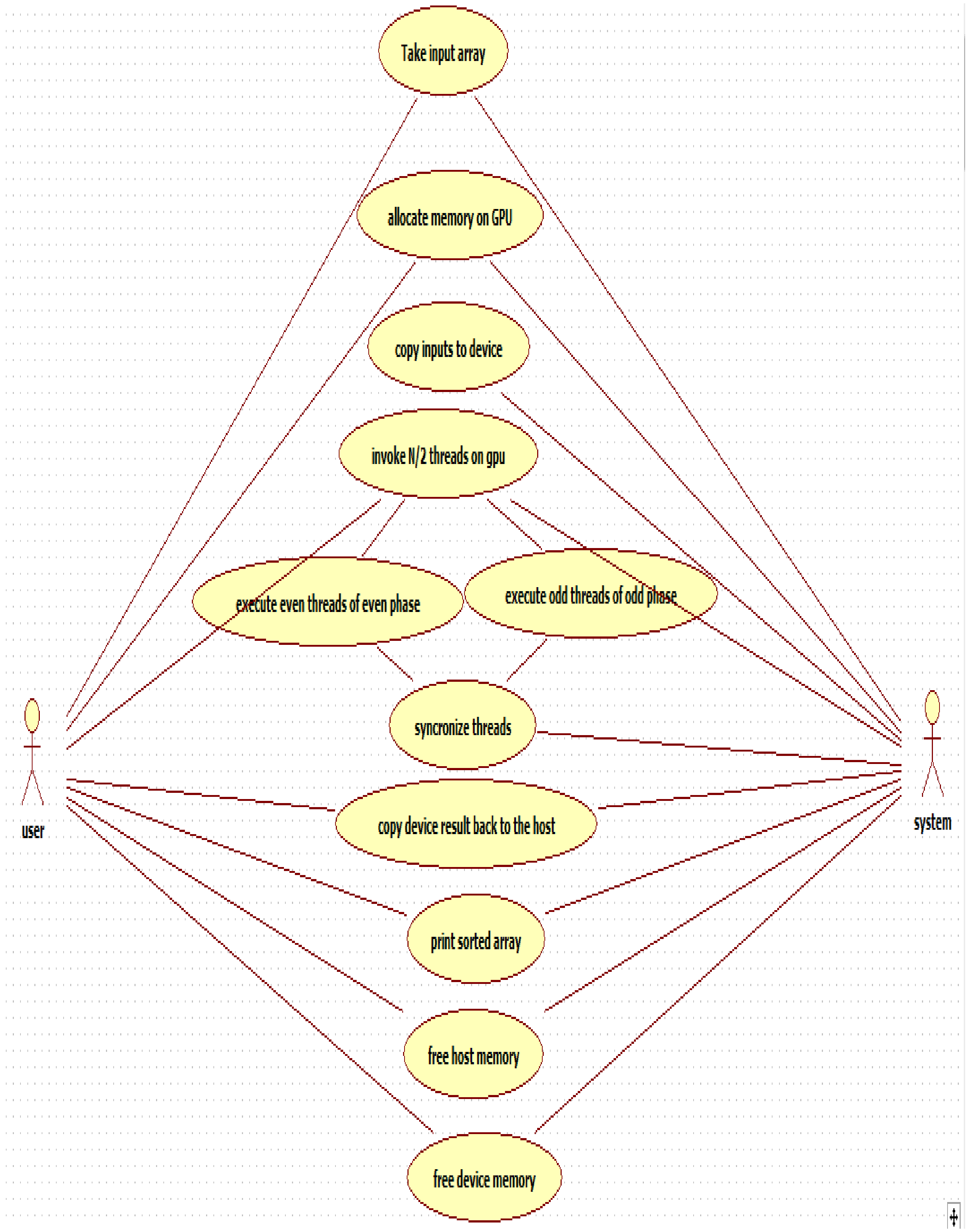
Negative Test cases

Test ID	**Test case Description**	**Actual Value**	**Expected Value**	**Result**
1	Check the division of a number by zero.	Error	Error	False
2	press = button	Error	Error	False
3	expression starts with * or /	**Error**	**Error**	**False**
4	Tan 90	**Invalid Input**	**Invalid Input**	

Assignment A_6 Encryption



Group B_2 Odd even sort



White Box Testing:-

White Box Testing White box testing is a testing technique, that examines the program structure and derives test data from the program logic/code. White Box Testing (WBT) is also known as Code-Based Testing or Structural Testing. White box testing is the software testing method in which internal structure is being known to tester who is going to test the software. White box testing is also known as clear box testing, open box testing, logic driven testing or path driven testing or structural testing and glass box testing. The White-box testing is one of the best method to find out the errors in the software application in early stage of software development life cycle.

What do you verify in White Box Testing?

In the White box testing following steps are executed to test the software code:

- ✓ Basically verify the security holes in the code.
- ✓ Verify the broken or incomplete paths in the code.
- ✓ Verify the flow of structure mention in the specification document
- ✓ Verify the Expected outputs
- ✓ Verify the all conditional loops in the code to check the complete functionality of the
- ✓ application.
- ✓ Verify the line by line or Section by Section in the code & cover the 100% testing.
- ✓ Above steps can be executed at the each steps of the STLC i.e. Unit Testing, Integration & System testing.

White Box Testing Techniques

Here are some white box testing techniques used in White Box Testing?

1. Statement Coverage: In this white box testing technique try to cover 100% statement coverage of the code, it means while testing the every possible statement in the code is executed at least once. Tools: To test the Statement Coverage the Cantata++ can be used as white box testing tool.

2. Decision Coverage: In this white box testing technique try to cover 100% decision coverage of the code, it means while testing the every possible decision conditions like if-else, for loop and other conditional loops in the code is executed at least once. Tools: To cover the Decision Coverage testing in the code the TCAT-PATH is used. This supports for the C, C++ and Java applications.

3. Condition Coverage: In this white box testing technique try to cover 100% Condition coverage of the code, it means while testing the every possible conditions in the code is executed at least once.

4. Decision/Condition Coverage: In this mixed type of white box testing technique try to cover 100% Decision/Condition coverage of the code, it means while testing the every possible Decisions/Conditions in the code is executed at least once.

5. Multiple Condition Coverage: In this type of testing we use to cover each entry point of the system to be execute once. In the actual development process developers are make use of the combination of techniques those are suitable for there software application.

How do you perform White Box Testing? Let take a simple website application. The end user is simply access the website, Login & Logout, this is very simple and day 2 days life example. As end users point of view user able to access the website from GUI but inside there are lots of things going on to check the internal things are going right or not, the white box testing method is used. To explain this we have to divide this part in two steps. This is all is being done when the tester is testing the application using White box testing techniques.

STEP 1) UNDERSTAND OF THE SOURCE CODE STEP 2) CREATE TEST CASES AND EXECUTE

STEP 1) UNDERSTAND OF THE SOURCE CODE:- The first & very important steps is to understand the source code of the application is being test. As tester should know about the internal structure of the code which helps to test the application. Better knowledge of Source code will helps to identify & write the important test case which causes the security issues & also helps to cover the 100% test coverage. Before doing this the tester should know the programming language what is being used in the developing the application. As security of application is primary objective of application so tester should aware of the security concerns of the project which help in testing. If the tester is aware of the security issues then he can prevents the security issues like attacks from hackers & users who are trying to inject the macious things in the application.

STEP 2) CREATE TEST CASES AND EXECUTE In the second step involves the actual writing of test cases based on Statement/Decision/Condition/Branch coverage & actual execution of test cases to cover the 100% testing coverage of the software application. The Tester will write the test cases by diving the applications by Statement/Decision/Condition/Branch wise. Other than this tester can include the trial and error testing, manual testing and using the Software testing tools as we covered in the previous article.

Types of White Box Testing:

1. Unit Testing
2. Integration Testing

1. Unit Testing What is a Unit?

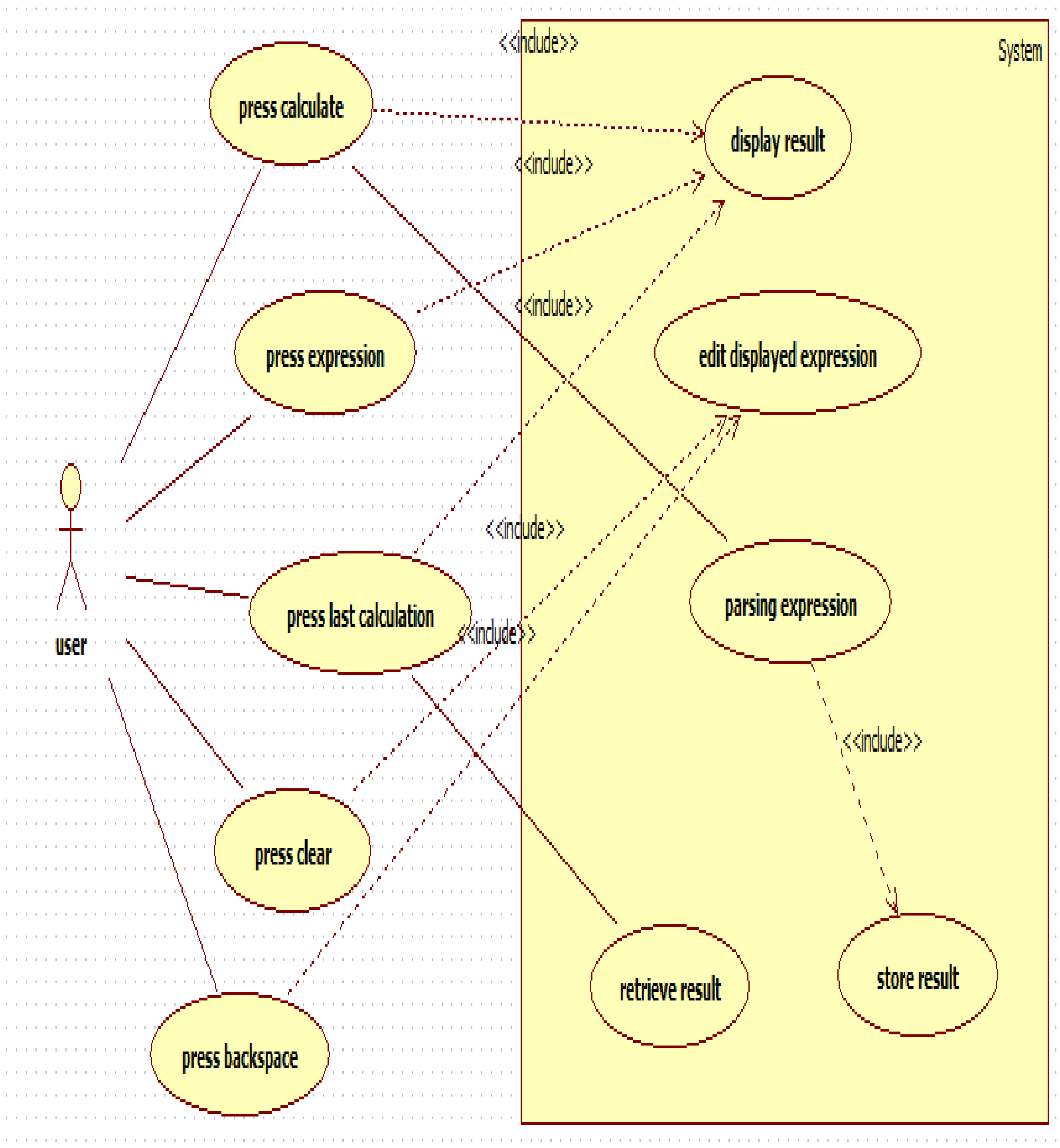
Unit is a single smallest independent component.
Unit Testing is categorized as:
1. Execution Testing
2.Operations Testing
3. Mutation Testing

2. Integration Testing: After making independent components, these components are joined together to check if they are working fine together as one application. Example: Compose Mail and Sent Items. After composing and sending mails, mails are checked in sent items. Different modules should inter talk with each other.

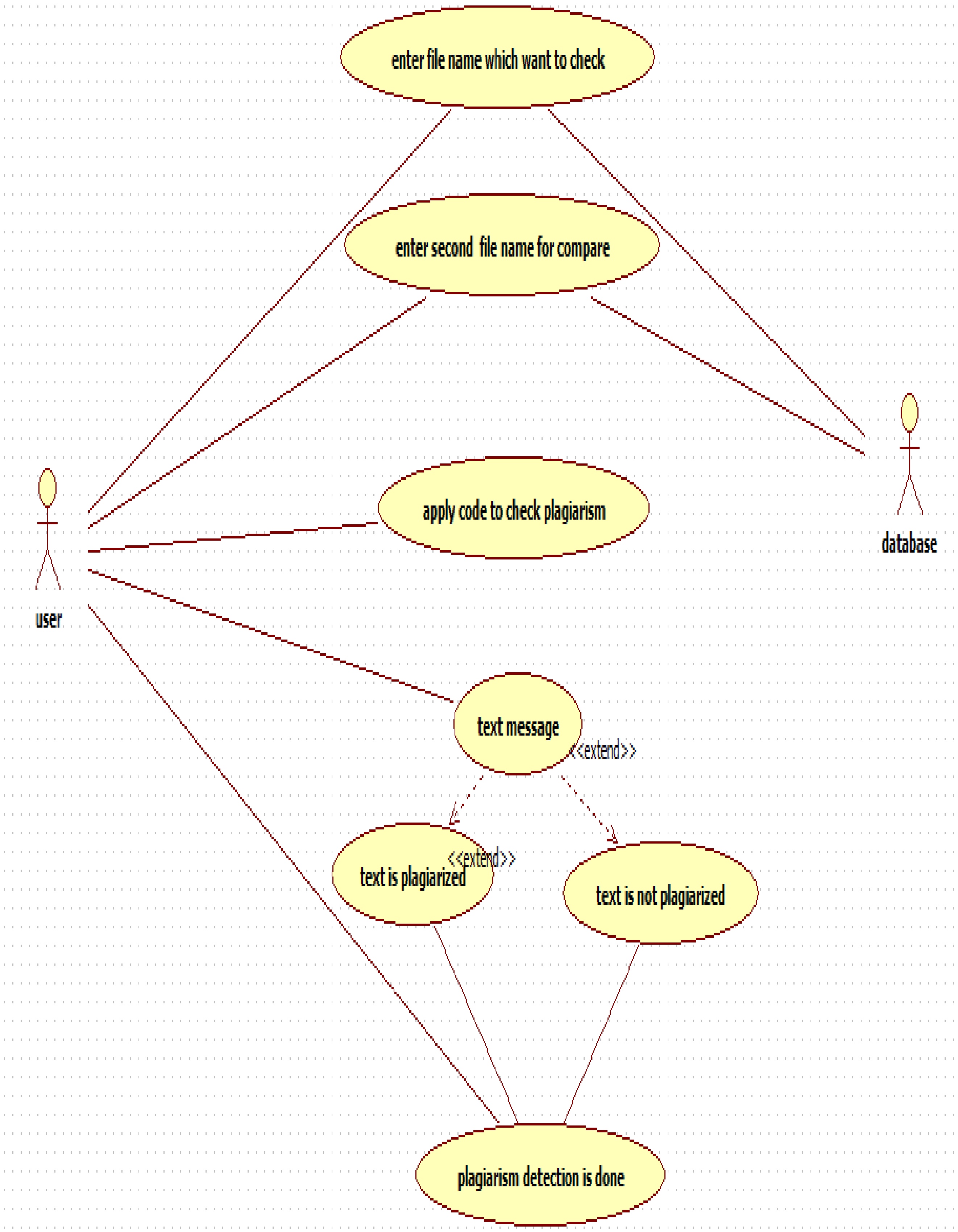
Approaches in Integration Testing:
1. Top Down Approach:
When main module is ready and sub module is under development. Main module is dependent on sub module. Here, stubs comes into picture. Stubs are temporary programs which takes the place of module under development. Ready module is tested using Stubs.

2. Bottom Up Approach: When Sub module is ready and main module is under development. Here, driver are the temporary programs which are used to test sub modules.

Group B_3 Calculator



Group B_4 Plagiarism checker


enter file name which want to check
enter second file name for compare
apply code to check plagiarism
user
database
text message
<<extend>>
<<extend>>
text is plagiarized
text is not plagiarized
plagiarism detection is done