|  |  |
| --- | --- |
| Assessment Title | Assignment (SAAirline Ticketing) |

## Competency Details

|  |  |
| --- | --- |
| Unit code/s and title/s | ICTPRG443 – Test software development |
| Qualification code/s and title/s | ICT40120 - Certificate IV in Information Technology |
| Business unit/Work group | BARTS/IT Studies |

## Instructions

|  |  |
| --- | --- |
| Method/s of assessment | Product (Create and Written) |
| Overview of assessment | This assessment will require you to conduct a unit testing for a given class library. Student will establish the test requirements, prepare a test plan, design the test cases, write the test scripts, execute the test scripts and record the test results. |
| Task/s to be assessed | This assessment will require you to complete the following tasks based on the SAAirlineTicketing scenario:   * Task 1 – Test requirements (team work & individual requirement report) * Task 2 – Test plan * Task 3 – Test case design * Task 4 – Test script and execution * Task 5 – Test results * Task 6 – Unit testing progress (team meeting and individual progress report) * Task 7 – Defect tracking |
| Time allowed | Refer to your schedule for submission dates |
| Location of assessment | Assessment can be completed anywhere with access to the resources required. (see Resources Required section below) |
| Decision making rules | To receive a satisfactory outcome for this assessment you must complete all parts correctly.  Word counts are provided as guidance only. |
| Assessment conditions | This assessment must be undertaken where the conditions replicate noise levels and interruptions that people typically experience working in the ICT industry.  This is unsupervised assessment and you may access any required resources.  The Task 1 involves team work and as well as individual work.  The Task 2 ~ 5 are individual work. |
| Resources required | To complete this assessment, you will require the following:   * Access to Learn with Internet access * Visual Studio * Code requiring testing – C# SAAirline Ticketing class library * Word processing software such as Microsoft Word. |
| Result notification and reassessment information | You will be provided feedback and the result for your assignment on TAFESA Learn. You will be and given the chance to resubmit with required corrections only once.  Refer to the TAFE SA assessment policy for more information <https://www.tafesa.edu.au/apply-enrol/before-starting/student-policies/assessment> |

**Assignment (SAAirline Ticketing)**

SAAirline provides airline service. They fly flights with economy or first class seats. A flight normally, for example, has economy class of 20 rows with each row has 8 seats across (i.e. 160 economy seats) and 5 rows with each row has 4 seats across (i.e. 20 first class seats).

Flight information and customer information are captured. Non-discount prices are set as:

First Class seat at $2,000.00

Economy class at $1,000.00

Customers are optional to join membership such as Global member or Asia Member. Some customers do not join membership at all. For those who joined, they can receive discount on their seats booked.

* GlobalWorld members receive 20% discount.
* AsiaWorld members receive 10% discount.
* Customers can book several seats. Seats availability is recorded. The system displayed the invoice of what the customer has booked.

The **SAAirline** system has an **Airline Ticketing application program** is developing in **C# class library** by their own software team and it is still in an incomplete state. The input and output screens are still in the developing stage and are not available for integration test nor user acceptance test. Those tests are out of the scope in this stage of development. The application only has several business classes in the class library which are available for testing. The SAAirline has contracted the **ITWorks Company Ltd**. to conduct **unit testing** of the **SAAirline Booking** **project** to ensure those business classes are working properly. You and your other team member are working as software testers in ITWorks. Your team has experience in using **NUnit testing framework**. Your testing team has a test manager called **Jake Brown**. The project starts from today. The contracted project is expected to be completed within 7 weeks. On week 5, the project is expected to be the milestone date for test progress report and test review.

The project has an urgency to complete due to the other development are lined up to be completed. Therefore any defect or failed test cases will have very high priority to get it fixed. All defect test cases must be redeveloped and re-tested with 2 weeks.

Given the following items:

* Visual Studio Sales System project with C# class library source codes (Student files)
* ITWorks Test Plan template - customize the details as appropriate (Student files)
* The dependency diagram (Appendix A)
* The class diagram (Appendix B)

**Assessment schedule:**

* Attend **test requirement team meeting** on **Session 2**.
* Submit **Task 1 (individual Test Requirements report)** to LEARN on **Session 3**.
* Attend **test progress team meeting** on **Session 5**.
* Submit **Task 6 (Unit Testing Progress Report)** to LEARN on **Session 5** using the **ITWorks Unit Testing Progress Report.docx** file.
* Submit the following items in a **zip** file to LEARN on **Session 7**:
* The C# Test Project application with the test driver & all references to your class library being tested.
* The Test plan (add/modify details from the template plus the test cases)
* The word document with your answers from TASK 2 ~ 5 & 7. You must include the screen shots of all test cases, the summary result of NUnit Project & the defect tracking report.

Complete the following tasks:

**Task 1. (Test Requirements – team meeting and individual requirement report)**

* Oral communication (team meeting) on **Session 2**.
* Test Requirements Report (individual work) due by **Session 3**.

You need to form a team with no more than two members. The team members will meet in a workshop to determine the testing requirements, standards and terms.

ITWorks expects team members to follow the communication protocols which are listed in **ITWorks Communication Protocols.docx**. While the team is conducting the meeting, your lecturer will act as the IT Manager and to sign off the team members has met those protocols specified as the following:

* Raise questions to seek for clarification in the context of test requirements. Each member raises at lease three questions
* Listen and respond to the question being asked by the other team members (each member respond at least three questions)

Your test manager (impersonate by your lecturer) will observe your meeting that you collaborate with your team members. After the meeting each team member will document the outcome of the meeting individually using a Word document by answering the following questions:

1. What kind of testing is required for the SAAirline project? Note the given software is incomplete. It only has the .dll file available for this testing project.
2. Name and describe two roles involved in the SAAirline Ticketing testing project? Describe each of these two roles with no more than 30 words.
3. What testing tools would you use for conducting the type of test that you have chosen in Part a.
4. What are the benefits of the testing type that you have identified in Part c above? (Approx. 30 words)
5. Name the standards being used by the testing industry for the following testing aspects:

* Testing Concepts and Definition
* Testing Documentation
* Unit testing

**Task 2 (Test Plan)**

Given the case scenario of the SAAirline Ticketing testing project, **develop a test plan**. For completing the test documentation, you must use the provided **ITWorks Test Plan template** which is given as part of the student file. Customize the test plan to suit for your SAAirline Ticketing unit test project. **Add, delete** or specify **N/A** (as not appropriate) on any section or item if it is not suitable to the project. Make sure that you delete the explanation which are in red colour. Replace your own content and change the fonts in black colour. Rename the file so that the file name does not have the word “template”.

**Task 3 (Test Case Design)**

In the following table list all classes and their methods in the class library.

1. For each of the class constructor and methods, what are the minimum number test cases that you will generate based on either:

Equivalence Partitioning

or Boundary value analysis

test case design techniques.

In your answer, other than the number of test cases for the methods, you must also state the test case design techniques being used.

**Task 3 (Test Case Design)**

In the following table list all classes and their methods in the class library.

1. For each of the class constructor and methods, what are the minimum number test cases that you will generate based on either:

Equivalence Partitioning

or Boundary value analysis

test case design techniques.

In your answer, other than the number of test cases for the methods, you must also state the test case design techniques being used.

|  |  |  |
| --- | --- | --- |
| Classes | Methods | No of test cases |
| Flight (e.g.) | **Constructor:**  Flight(int economyRows, int economySeats, int firstClassRows, int firstClassSeats) i.e. possible integers -ve, 0, +ve è 3\*3\*3\*3= 81  **Using Equivalence Partitioning**  **Methods of the class:**  getFirstClass() **Using Equivalence Partitioning**  getEconomy() **Using Equivalence Partitioning**  setEconomy(ArrayList economy) **Using Equivalence Partitioning**  setFirstClass(ArrayList firstClass) **Using Equivalence Partitioning** | 81  1  1  1  1 |
| Activity | **Constructor:** Activity(Flight theFlight) **Using Equivalence Partitioning**   |  | | --- | |  |  |  | | --- | | bookSeats(int priceCode, int number, Customer theCustomer) **Using Equivalence Partitioning** |   getCustomerBooking(Customer cust) - **Using Equivalence Partitioning** | 2  18  2 |
| AsiaWorld  Member | getdiscount() **Using Equivalence Partitioning** | 2 |
| GlobalWorld  Member | getdiscount() **Using Equivalence Partitioning** | 2 |
| Customer | **Constructor:** Customer(int memberType, String firstName, String lastName, String creditNumber, String creditType, String expiry)) - **Using Equivalence Partitioning**  getCreditNumber() **Using Equivalence Partitioning**  getLastName() **Using Equivalence Partitioning**  getFirstName() **Using Equivalence Partitioning**  getCreditType() **Using Equivalence Partitioning**  getExpiry() **Using Equivalence Partitioning**  getMemberType() **Using Equivalence Partitioning**  setCreditNumber(String creditNumber) **Using Equivalence Partitioning**  setCreditType(String creditType) **Using Equivalence Partitioning**    setLastName(String lastName) **Using Equivalence Partitioning**    setFirstName(String firstName) **Using Equivalence Partitioning**  setExpiry(String expiry) **Using Equivalence Partitioning**  setMemberType(int memberType) **Using Equivalence Partitioning** | 96  1  1  1  1  1  1  2  2  2  2  2  3 |
| Invoice | **Constructor:** Invoice(int priceCode, Customer theCust, int rowNum, int startSeatNum, int seatsBooked) - **Using Equivalence Partitioning**  getNumberOfSeats()**Using Equivalence Partitioning**  getPriceCode() **Using Equivalence Partitioning**  getTheCust() **Using Equivalence Partitioning**  getStartSeatNum() **Using Equivalence Partitioning**  getRowNum() **Using Equivalence Partitioning**  getDiscount(int memberCode) **Using Equivalence Partitioning**  getCharge(int priceCode) **Using Equivalence Partitioning**  setNumberOfSeats(int numberOfSeats) **Using Equivalence Partitioning**  setPriceCode(int priceCode) **Using Equivalence Partitioning**  setTheCust(Customer theCust) **Using Equivalence Partitioning**  setRowNum(int rowNum) **Using Equivalence Partitioning**  setStartSeatNum(int startSeatNum) **Using Equivalence Partitioning** | 162  1  1  1  1  1  3  3  3  3  2  3  3 |
| Seat | **Constructor:** Seat(int numAvail, int code) **Using Equivalence Partitioning**  bookSeats(int num) **Using Equivalence Partitioning**  getPriceCode() **Using Equivalence Partitioning**  getLastBooked() **Using Equivalence Partitioning**  getCurrentSeat() **Using Equivalence Partitioning** | 9  3  2  2  2 |
|  | Total no of test cases: | 431 |

1. Since there are too many test cases to fully test the application, you are only required to design 10 test cases in total.

For the 10 test cases, you must include:

* Four test cases must be related with the methods from the invoice class, Seat class or Activity class. (e.g. the calculations of seats, discount & price code).

Methods from the **Invoice** class:

* setNumberOfSeats(int numberOfSeats)
* setPriceCode(int priceCode)
* setTheCust(Customer theCust)
* setRowNum(int rowNum)
* setStartSeatNum(int startSeatNum)
* getDiscount(int memberCode) e.g. *Asia World or Global world members with correct discount*
* getCharge(int priceCode)

Methods from the **Seat** class:

* bookSeats(int num)

Method from **Activity** class:

* bookSeats (int priceCode, int number, Customer theCustomer)
* getCustomerBooking(Customer cust)
* One test case on null (e.g. IsNull) value. e.g. no customer name
* One test case on same object (i.e. AreSame) *e.g. Test the Invoice in getting a customer (Create a customer & invoice objects, set customer to existing object. i.e. Sales Customer)*
* Two test cases to test for exceptions. (e.g. argument exception, argumentNullException, argumentOutOfRangeException etc.)
* Two others as you prefer.

Develop unit test cases and document the test case design in the following table. For each test case, fill the following information:

* Test case id – all test cases must be numbered.
* Test case description – summarise the test case with a few words.
* Reason to test in the business context – explain why you want to do the test.
* Data input – what object need to construct and what input parameters used.
* Test case technique used (e.g. specify boundary value analysis or equivalence class partition).
* Expected result – The tester calculates the expected result according to business process.
* Date/Time/Duration – The date and time the test is carried out and how long. The duration includes time spent the planning, design and execution.
* Resources required – include tester, test manager or product owner or delegate.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Test Cases Design** | | | | | | | |
| **Test case id**  **TC xxx** | **Name of method being tested e.g. class.method().**  **No need to include the parameters in this column.** | **Reason to test in business context** | **Data input (Constructor used and the Method to be tested include the parameters)** | **Test Case Design Technique used** | **Date/time**  **&**  **Duration (hours)** | **Resources Required**  **(Name of tester, hardware & software needed)** | **Expected result** |
| TC 1 | Invoice.setNumberOfSeats() | Validate that the method accurately sets the number of seats for the invoice. | Invoice object, setNumberOfSeats(4) | Equivalence Partitioning | 15/11/2024 (1 hr) | Tester, Visual Studio with NUnit | Number of seats in the invoice is set to 4. |
| TC 2 | Invoice.setPriceCode() | Ensure that the price code is set correctly to calculate pricing according to the customer’s selected seat type. | Invoice object, setPriceCode(2) (Economy) | Equivalence Partitioning | 15/11/2024 (1 hr) | Tester, Visual Studio with NUnit | Price code for the invoice is set to 2 (Economy class). |
| TC 3 | Invoice.getDiscount() | Verify that the discount for members (AsiaWorld, GlobalWorld) is calculated accurately. | Invoice object, getDiscount(1) (AsiaWorld) | Equivalence Partitioning | 15/11/2024 (1 hr) | Tester, Visual Studio with NUnit | Returns 10% discount for AsiaWorld members. |
| TC4 | Invoice.getCharge() | Confirm that the charge is calculated correctly based on the price code for the seat class. | Invoice object, getCharge(1) (First Class) | Equivalence Partitioning | 15/11/2024 (1 hr) | Tester, Visual Studio with NUnit | Returns charge of 2000.0 for First Class. |
| TC 5 | Seat.bookSeats() | Confirm that seat booking functionality works as expected by reducing available seats accordingly. | Seat object, bookSeats(3) | Equivalence Partitioning | 15/11/2024 (1 hr) | Tester, Visual Studio with NUnit | Reduces available seats by 3 and confirms booking. |
| TC 6 | Activity.bookSeats() | Verify seat booking through Activity class, ensuring integration with pricing and customer booking. | Activity object, bookSeats(2, 4, Customer object) | Equivalence Partitioning | 15/11/2024 (1 hr) | Tester, Visual Studio with NUnit | Seats are booked and associated with the customer in the system. |
| TC 7 | Customer.Name (null test) | Test if the application handles null values appropriately for essential customer details like name. | Customer object, Name = null | Equivalence Partitioning | 15/11/2024 (1 hr) | Tester, Visual Studio with NUnit | Throws an exception or error when Name is null, as name is required. |
| TC 8 | Invoice.setTheCust() | Test if an existing customer object can be assigned to an invoice, validating AreSame logic. | Invoice object, setTheCust(Customer object) (pre-existing object) | Equivalence Partitioning | 15/11/2024 (1 hr) | Tester, Visual Studio with NUnit | Invoice’s customer matches the pre-existing Customer object assigned. |
| TC 9 | Seat.bookSeats() | Test booking for a seat number outside valid bounds to check for ArgumentOutOfRangeException. | Seat object, bookSeats(-1) | Equivalence Partitioning | 15/11/2024 (1 hr) | Tester, Visual Studio with NUnit | Throws ArgumentOutOfRangeException when booking seats with a negative number. |
| TC 10 | Invoice.setRowNum() | Validate that the setRowNum() method handles an invalid row number correctly, using exception handling to prevent improper input. | Invoice object, setRowNum(-1) | Equivalence Partitioning | 15/11/2024 (1 hr) | Tester, Visual Studio with NUnit | Throws ArgumentOutOfRangeException for negative row numbers. |

**TASK 4. (Test Script and Execution)**

1. In the Visual Studio, create a test driver to test the class library using the NUnit test framework. Implement the unit tests. For each test cases designed in TASK 3, you are required to implement the algorithm in the test scripts/procedures (i.e. construct the appropriate objects with the designed test input parameters). You need to work out the manually for the expected results.
2. Execute your test cases. Submit a screen shot that all test cases have been executed.

A screenshot of a computer

Description automatically generated

**TASK 5. (Test Results)**

**Test Cases Report.** Execute the test cases if they have completed the test case designed. Record the test results or state the current status after the execution of the test cases. Record the Actual Result and specify whether the test is “Pass” or “Fail”.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Cases Result as at: dd-mm-yyyy** | | | | **Test Case Result:**  **Fill these only after the execution of the tests** | |
| **Test case id**  **TC xxx** | **Name of method being tested e.g. class.method()** | **Data input (include the parameters)** | **Expected result** | **Actual result** | **Pass / Fail or Not executed yet** |
| TC 1 | Invoice.setNumberOfSeats() | 4 | The number of seats should be set to 4. | The number of seats was set to 4. | Pass |
| TC 2 | Invoice.setPriceCode() | Seat.FirstClass | The price code should be set to First Class. | The price code was set to First Class. | Pass |
| TC 3 | Invoice.getDiscount() | 0.9 | AsiaWorld members should receive a 10% discount (0.9 multiplier). | AsiaWorld members received a 10% discount (0.9 multiplier). | Pass |
| TC 4 | Invoice.getCharge() | 2000.0 | Expected charge for First Class is 2000.0. | charge for First Class was 2000.0. | Pass |
| TC 5 | Seat.bookSeats() | 3 | 4 (Last booked seat number should be updated to 4 after booking 3 seats). | Last booked seat number should be updated to 4 after booking 3 seats. Expected: 4 But was: 1 | Fail |
| TC 6 | Activity.bookSeats() | Seat.Economy, 2, testCustomer | Invoice should reflect the number of seats booked | Invoice reflected the number of seats booked | Pass |
| TC 7 | Customer Constructor (Null Test) | Customer.AsiaWorld, null, "Doe", "123456789", "Visa", "12/24" | <System.ArgumentNullException> | ArgumentNullException. | Pass |
| TC 8 | Invoice.setTheCust() | testCustomer | Customer object in Invoice should refer to the same test customer instance. | Customer object in Invoice refered to the same test customer instance. | Pass |
| TC 9 | Seat.bookSeats() (Exception Test) | 11 | ArgumentOutOfRangeException | ArgumentOutOfRangeException. | Pass |
| TC 10 | Invoice.setRowNum() | 6 | Row number should be set to 6 | Row number was set to 6 | Pass |

Note: Copy test cases from **Task 4** and fill the following columns:

* **Actual result** - result must be returned from the execution of the method.
* **Pass or Fail** – after the test execution, the test result must be either pass or fail.
* **Not Executed yet** – These test cases are behind in the test schedule.

For submission:

Capture one screen shot for each test cases that are failed in the test execution. e.g. if you have two test cases failed, you need to screen shots. In each screen shot, you must show both the expected result and actual result. Note: The expected result is in the test script. The actual result is in the Test Detail Summary of the Test Explorer.

**Test Case 5:**

A white screen with black text

Description automatically generated

**TASK 6 (Unit Testing Progress - Team Meeting and Individual Progress Report)**

**Note:**

* The team meets your test manager (i.e. your lecturer) on **Session 5** with the progress reports.
* This task must be carried out on **Session 5** disregard whether the test cases have fully or partially completed.
* Individual members fill the **ITWorks Unit Testing Progress Report.docx** document in part a to d and make it available during the meeting. You also upload the file to LEARN for marking.
* Review the test report & test progress report with the test manager. The test manager will discuss the status options with you.

Other than meeting the test manager, you are required to produce a progress report by submitting the **ITWorks Unit Testing Progress Report.docx** document which can be found in the student file. You need to submit the file to LEARN for marking on Session 5.

The following table only shows what need to be done in this task.

1. **Progress indicator**. You are required to report the progress of the test project. Just record the number of test cases are completed on Session 5. Complete a test summary which analyses the test results by filling the following table.

|  |  |
| --- | --- |
| **Progress indicator as at: dd-mm-yyyy**  **(Please enter the date of the Session 5 above)** | |
| Total No of Test Cases (for the project) | 10 |
| Test Cases completed the designed at this stage | 10 |
| Test Cases executed | 10 |
| Test Cases - Pass | 7 |
| Test Cases - Fail | 3 |
| Test Cases Not Executed | 0 |

Capture a screen shot in the space below to show the summary of the test result of all tests run from the NUnit project.

A screenshot of a computer

Description automatically generated

1. **Test Progress Report.** Just record what test cases are done and what test cases are not done on Session 5. Fill the following two tables:

\*\*Note: Use the date of Session 1 as the “report from date”

Use the date of Session 5 as the “report to date” and the “review date”.

|  |  |
| --- | --- |
| Project Name: | SAAirline Ticketing |
| Application Name: | SAAirline Ticketing |
| Report Date (From: dd-mmm-yyyy) | 30/10/2024 |
| Report Date (To: dd-mm-yyyy) | 12/11/2024 |
| Completed by (milestone date for the unit testing review date): | 12/11/2024 |
| Test Manager: | Jake Brown |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Unit / Functional Testing** | | | | |  |
| **Test Case ID** | **Description** | **Test Date** | **Tester** | **Status**  **(See possible status below)** | **Comment** |
| TC 1 | Invoice.setNumberOfSeats() | 12/11/2024 | Travis Griggs | Passed |  |
| TC 2 | Invoice.setPriceCode() | 12/11/2024 | Travis Griggs | Passed |  |
| TC 3 | Invoice.getDiscount() | 12/11/2024 | Travis Griggs | Passed |  |
| TC 4 | Invoice.getCharge() | 12/11/2024 | Travis Griggs | Passed |  |
| TC 5 | Seat.bookSeats() | 12/11/2024 | Travis Griggs | Fail | Last booked seat number should be updated to 4 after booking 3 seats. Expected: 4 But was: 1 |
| TC 6 | Activity.bookSeats() | 12/11/2024 | Travis Griggs | Passed |  |
| TC 7 | Customer Constructor (Null Test) | 12/11/2024 | Travis Griggs | Passed |  |
| TC 8 | Invoice.setTheCust() | 12/11/2024 | Travis Griggs | Passed |  |
| TC 9 | Seat.bookSeats() (Exception Test) | 12/11/2024 | Travis Griggs | Passed |  |
| TC10 | Invoice.setRowNum() | 12/11/2024 | Travis Griggs | Passed |  |

Note: The test date is the date that the test case is being conducted.

**Possible Status:**

**Not run** - Initial status for a new test case. Indicates that the test case has been created but is not yet executed.

**In progress** - Indicates that the test case is being run, but not yet completed.

**Passes** -  Indicates that the test case was executed and met the expected results.

**Failed** - Indicates that the test case did not meet the expected result. It will trigger a defect update.

**Blocked** - Indicates that the test case cannot be run for some reason.

**Retest** - Indicates that the test case is going to be retested. (i.e. failed test case become ready for execution again)

**Deferred** – Use this to closed out a failed, blocked or not run testcase due to test effort has run out of time. Test management decided to accept the risk of implementing an untested feature.

**Task 7. (Defect Tracking)**

Fill the following table for the test cases that are failed in your test cases. If you do not have a fail test case, generate one deliberately, therefore it is testing to fail.

Defect id (Dxxx) is related with the test case id (TCxxx). e.g. D5 is related with TC5.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Defect ID    D xxx | Priority  (low 1 ~ high 3) | Severity  (low 1 ~ high 3) | Created by team member | Created Date | Team member Assigned to | Expected Resolved Date (Exp. in two weeks) | Status (See possible options below) |
| D5 | 3 | 3 | Travis Griggs | 22/11/24 | Travis Griggs | 5/12/24 | **Need more information** – The defected code may have complex logic that need more information to clarify the algorithm of the unit code. |

Provide a screen shot for each failed test case (i.e. Shown the test case result) that needs tracking.

**Defect D5:**

A white screen with black text

Description automatically generated

**Note:**

**Created date:** use the data in session 7

**Possible Status:**

**New** - Initial status for a new test case. Indicates that the test case has been created but is not yet executed.

**Fixed** – The defect code has been fixed by the development team and open for testing.

**Open** – A test case has been designed and approved for testing.

**Reopen** – A test case may have been closed or blocked and re-stated to execute the test case again.

**Duplicate** -  Add test cases with the same basic information, save time by duplicating an existing test case and then editing the new test case.

**Invalid** - Write a negative test case by selecting the invalid input data or undesired user behaviour that you acquire by performing positive test case steps.

**Won’t fix** – Due to requirement may have changed, the test case no longer contribute to the part of the integrated code any more.

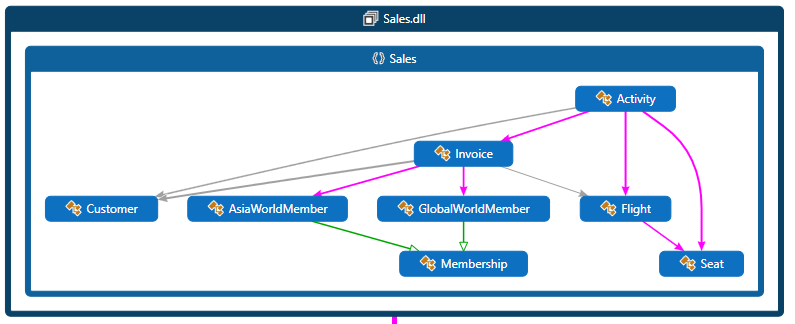
**Need more information** – The defected code may have complex logic that need more information to clarify the algorithm of the unit code.

**Suspended** – The test case is suspended due to a portion of testing activities are suspended e.g. critical path deadline is met and no more delivery is acceptable or holiday shuts down for both development and testing.

**Closed** – A test case is closed when there will be no more work on it.

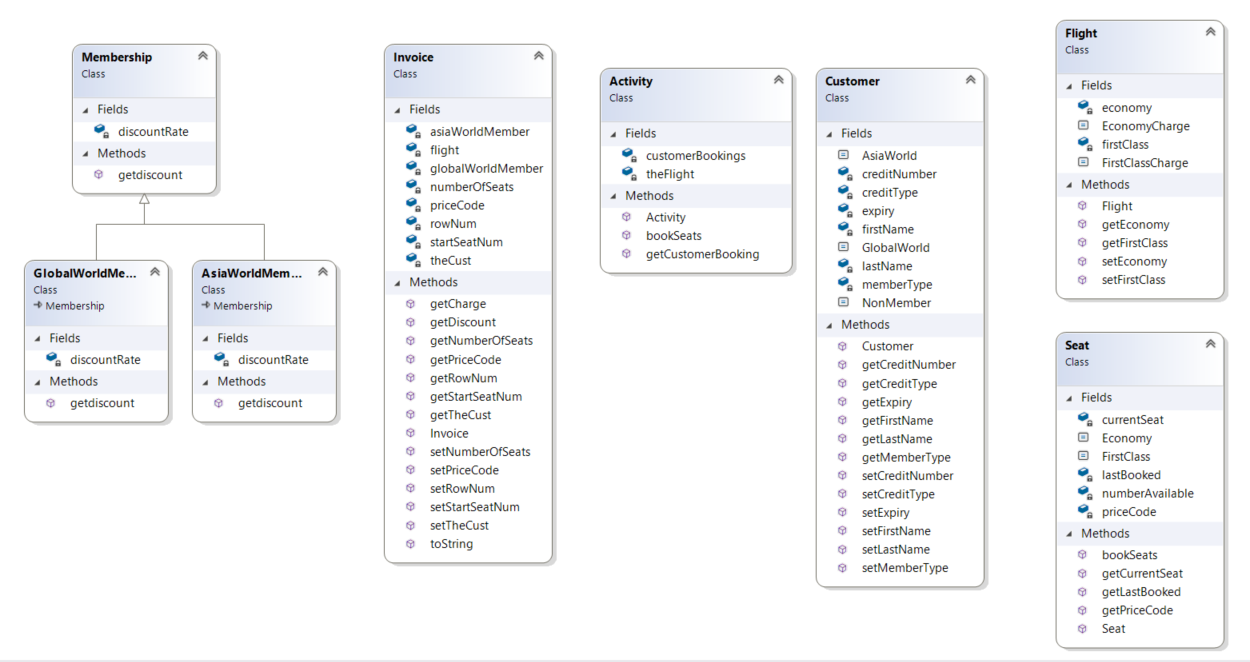
**Appendix A:**

**Class dependency diagram:**



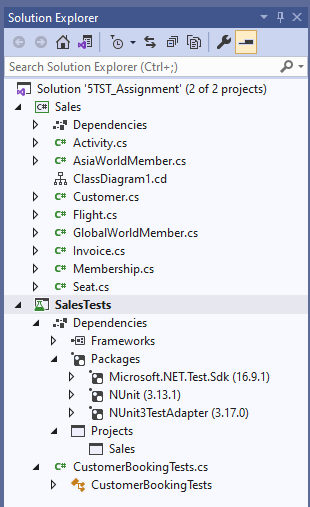
**Appendix B.**

**Class Diagram:**

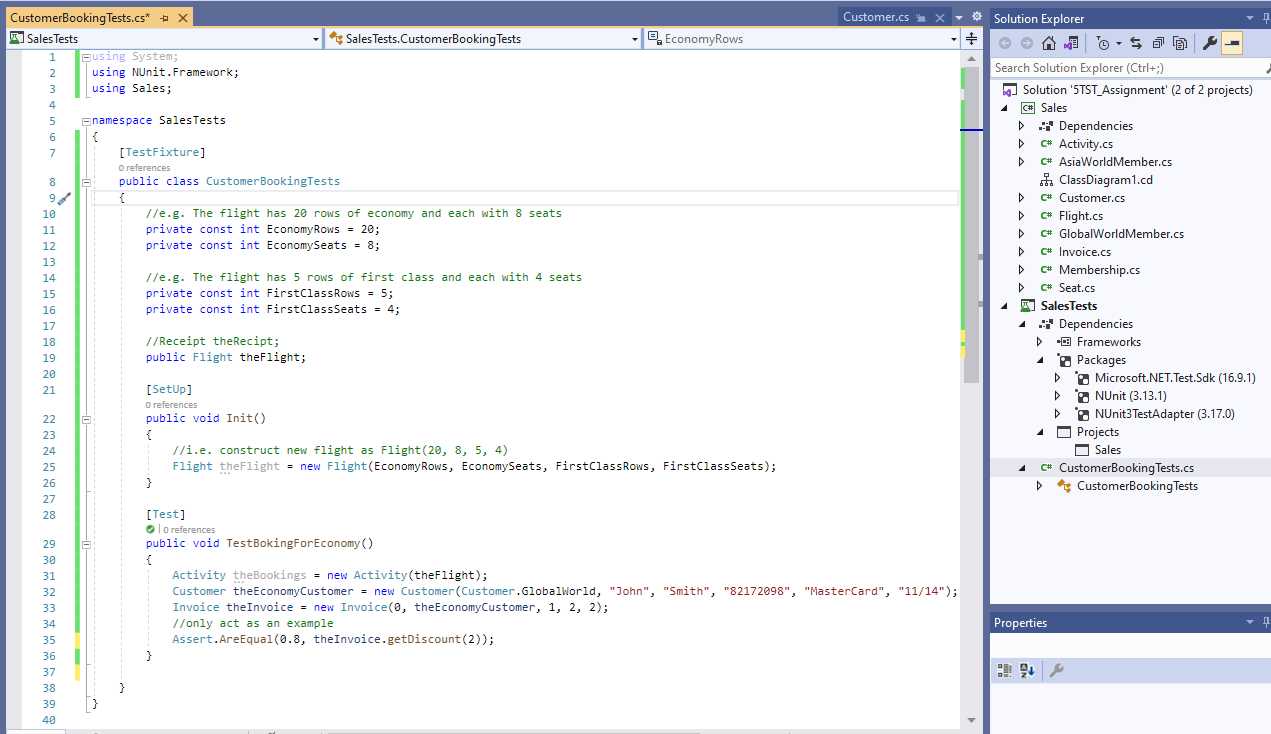


**Appendix C:**

**Showed project references and reference packages:**



**A test fixture example:**



Sample of an execution of a test case.