

Assignment

by William Baker Oliver

Submission date: 02-May-2025 10:35AM (UTC+0100)

Submission ID: 257572948

File name: 50534_William_Baker_Oliver_Assignment_648183_82603745.docx (1.25M)

Word count: 5413

Character count: 28498

Software Engineering Principles and Testing

Assignment: Software Engineering Design & Testing

William Baker Oliver

Student Number: 2314716

Computing (Games Development)

Table of Contents

Introduction 3

1: Evaluation of Software Design 4

2: User Stories, User Stories Map and Benefits 11

3: CRC Card Design Approach 14

4: Software Testing Evaluation 18

5: Application and Investigation of Unit Testing 25

Conclusion 26

Appendix 26

Bibliography 27

Introduction

This assignment covers a wide range of topics and methodologies regarding software engineering, testing, and designing. The software design for a Connect 4-style game is evaluated, in addition to the user stories and their benefits, as well as evaluating the CRC card approach and the cards produced for the design. This assignment also covers the importance of software testing and testable code, including showing the differences between black box and white box testing by providing examples that could be used for the design. Finally, unit testing and its principles are covered, including how unit testing can be applied to white box tests.

1: Evaluation of Software Design

One of the tasks for this assignment was to create a software design for a web application: a Connect 4-style game, which was to be done in a team. Much of the design work was created on Miro [1], a digital collaboration board where users could add to and edit boards, making it a perfect tool for the group work.

For the design, a UML diagram was created. The Unified Modelling Language, also known as UML, is a “a family of graphical notations, backed by single meta-model, that help in describing and designing software systems, particularly software systems built using the object-oriented (OO) style”, according to Martin Fowler and Kendall Scott [2]. UML essentially is a modelling language which allows users to create visualizations of software in blueprint form, showing the structure and relationship between classes, functions, as well as other different components in the software.

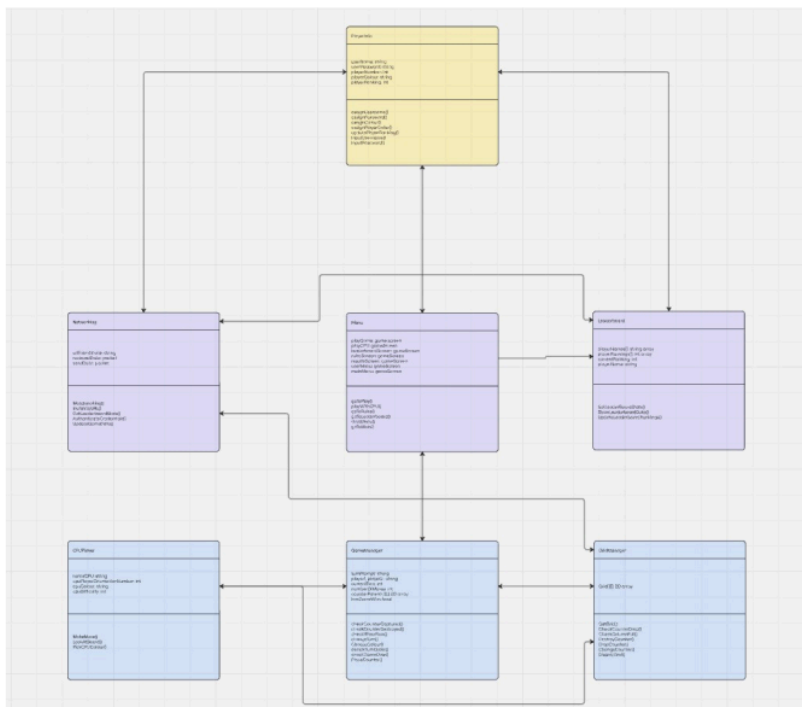


Fig.1: A screenshot showing the UML design for the Connect 4-style game. Created with Miro.

This diagram shows an example structure for the Connect 4-style game, complete with classes, attributes, objects and functions, with arrows indicating the relationship between classes and functions. The team worked on separate classes each, designating a potentially important area of the software. Take the PlayerInfo class below, it contains the class name at the top, the attributes in the middle, and the functions at the bottom. It contains the potential information the class may need, including the name of the user, their password, their colour when playing Connect 4 to change the look of the counters, etc. The class also has arrows pointing to which other classes it interacts with, such as the Networking class and the Leaderboard class.

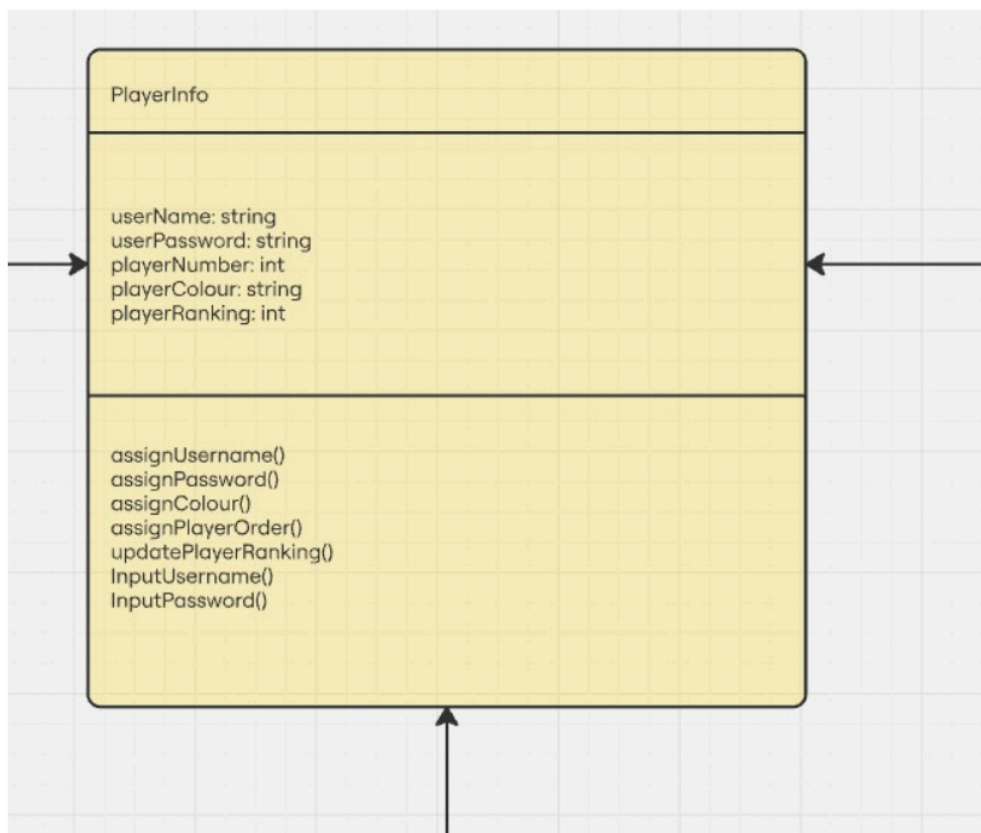


Fig.2: A screenshot showing the PlayerInfo class on the UML design for the Connect 4-style game. Created with Miro.

In addition to the UML diagram, a UML sequence diagram was also created. A UML sequence diagram essentially describes how operations will carry out in the program over time [3]. Shown below is a diagram of the Connect 4-style game, with each class that is involved being represented at the top, with the arrows designating functions and actions occurring over time, showing how they communicate with each other.

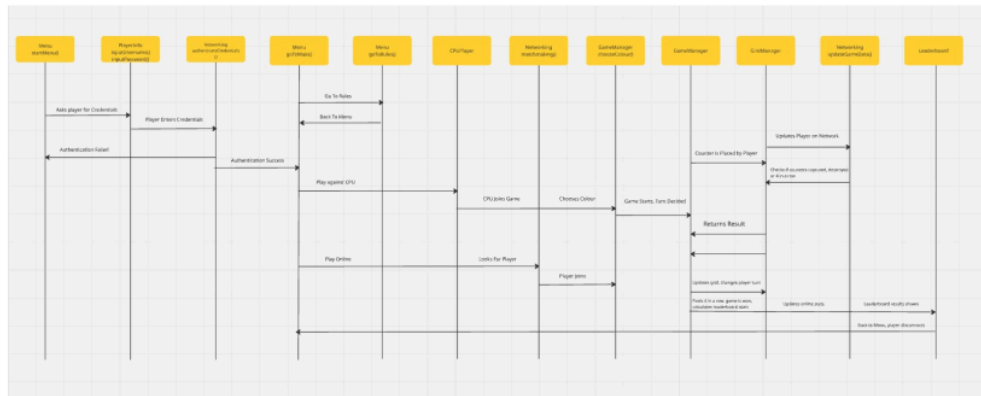


Fig.3: A screenshot showing the UML Sequence graph created for the Connect 4-style game. Created with Miro.

On reflection, these diagrams do an effective job of showing the potential structure classes and functions in the software, in addition to providing a useful visual for how the game flow and core functionality will work. However, there are weaknesses to these diagram approaches. The UML diagram lacks detail in certain areas, such as the visibility of certain attributes or functions, whether they be private or public for example. The UML sequence diagram, whilst doing an effective job of showcasing the workings of the software, is also basic in design, and some arrows might not have a clear enough indication of what they are doing.

In addition to the UML Sequence Diagram, the team produced wireframe diagrams. Wireframe diagrams act as a rough blueprint for any software or application, to help map out the basic structure, acting as a very simple sketch for the layout of the software [4].

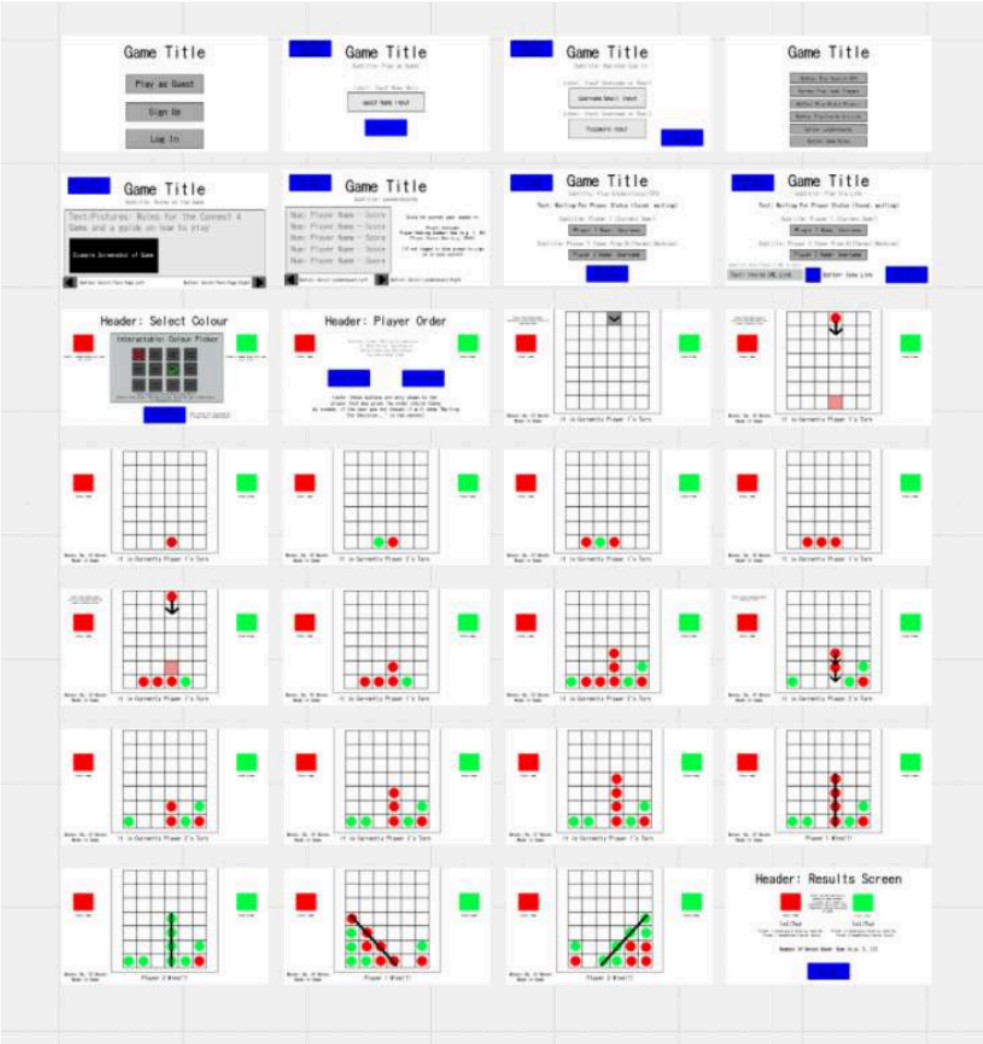


Fig.4: A screenshot showing the Wireframe diagrams created for the Connect 4-style game. Created with Miro.

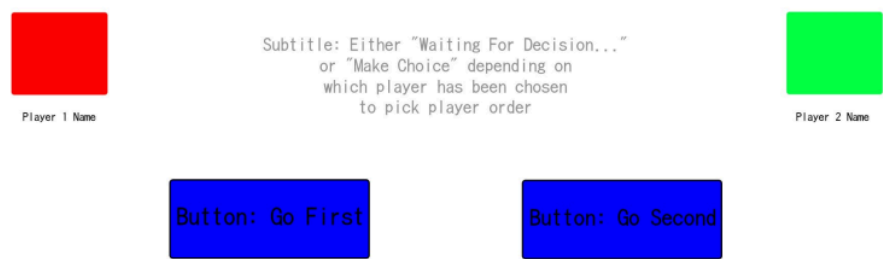
The team created multiple wireframe diagrams for the game, which showcase a rough look for how the menu and UI for the game will potentially look, with basic shapes acting as buttons, and text having labels along with notes describing certain elements of the design. It helps visualize the basic core of the game for the designers, the developers, and the stakeholders, the users of the software. The team were able to create a variety of diagrams, which helps communicate the potential vision for the software to others.



Fig.5: A screenshot of a wireframe diagram showing the design for the game rules menu.

Shown above is a wireframe diagram for the game rules menu. It shows where buttons will be placed on the menu, along with where the text and accompanying screenshots for the rules will go. On the top, it shows where the title for the game will go, along with a subtitle which helps designate the current menu that the player is on. It is a basic layout; however, it gives a good and clear sense of what the team envision the layout to look like.

Header: Player Order



(note: these buttons are only shown to the player that was given the order choice (done by random, if the user was not chosen it will show "Waiting For Decision..." in the center)

Fig.6: A screenshot of a wireframe diagram showing the player order screen before the game starts, showing a note that describes how the screen changes depending on who was chosen.

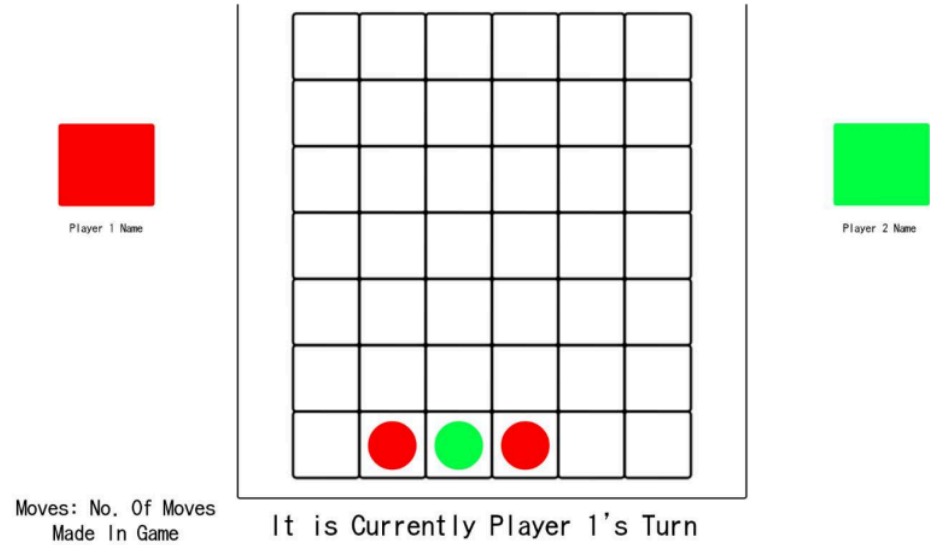


Fig.7: A screenshot of a wireframe diagram showing the game in action, with 2 counters surrounding the other player's counter.

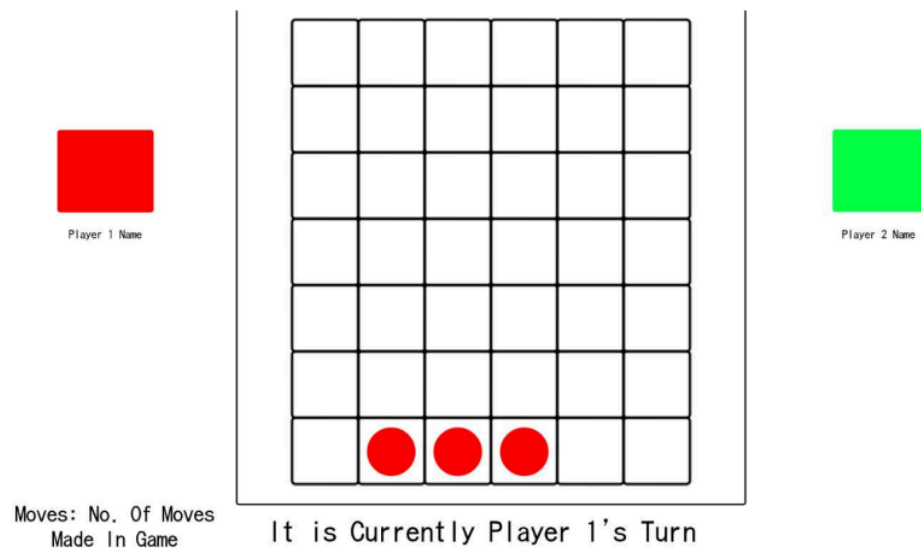


Fig.8: A screenshot of a wireframe diagram showing the game in action, showing the aftermath of Player 1's turn, where he has captured Player 2's piece.

This is another example of a wireframe design for the game, showing 2 images of the main game in action. This shows off a strength of the team's wireframe designs, as this can help show how the extra capture rules work for the game, as it shows when 2 counters from another player surround one counter from the other player, it transforms it into their counter. This is a great way to convey how the game works to other people who may not understand the concept, or the rules proposed for the game.

The designs produced by the team were solid and had a clear vision for the game. However, these diagrams use colour to demonstrate the colour changing for the players and their counters to differentiate themselves. Typically, a wireframe diagram does not use colour, only basic shades, so that the design or development team only focus on the layout and functionality of the software, rather than the looks. In addition, extra notes on certain diagrams would prove useful for describing the game's layout and its functionality, such as with the diagrams showing a counter being taken or destroyed. Whilst it looks fine on the surface, there is always the potential for important details to get lost in translation during the development process. Overall, however, the wireframe designs were a good effort by the team and is one of the strongest points of the overall design.

2: User Stories, User Stories Map and Benefits

In the process of creating the software design for the Connect 4 game, certain user stories were produced within the team on a user story map. According to Mike Cohn, user stories describe “functionality that will be valuable to either a user or purchaser of a system or software” [5]. User Stories focus more on functionality that will be useful to the potential users or customers of the software, rather than the technical aspects such as being written in a specific programming language.

As many software developers will only be focused on the technical workings and the code in the software, they may neglect and overlook the needs of the user in the development process. This is the benefit of user stories: they describe what the user wants to do with the software from their perspective, putting the focus exactly on the user’s needs. This gives both the software developers and designers a clearer direction on what needs to be implemented.

When creating user stories, many teams choose to use what is known as a User Story Map. According to Mike Cohn, a user story map is a “two-dimensional visual representation of actions a user will perform to achieve some goal.” [6]. This method was created by Jeff Patton, who states that “Story mapping keeps us focused on users and their experience, and the result is a better conversation, and ultimately a better product.” [7].

A map will typically contain user activities and tasks at the top, such as the user opening the software or the user entering an input into the software. These form the step-by-step process of the story map, which will eventually lead to a goal of sorts. Underneath are the user stories themselves, with each user story being tailored to the row’s user activities/tasks. Each story is placed on a ranking of most important to least important.

For the Connect 4-style game, the team created multiple user stories and placed them on a user story map board.

William Baker Oliver: Software Engineering Principles and Testing

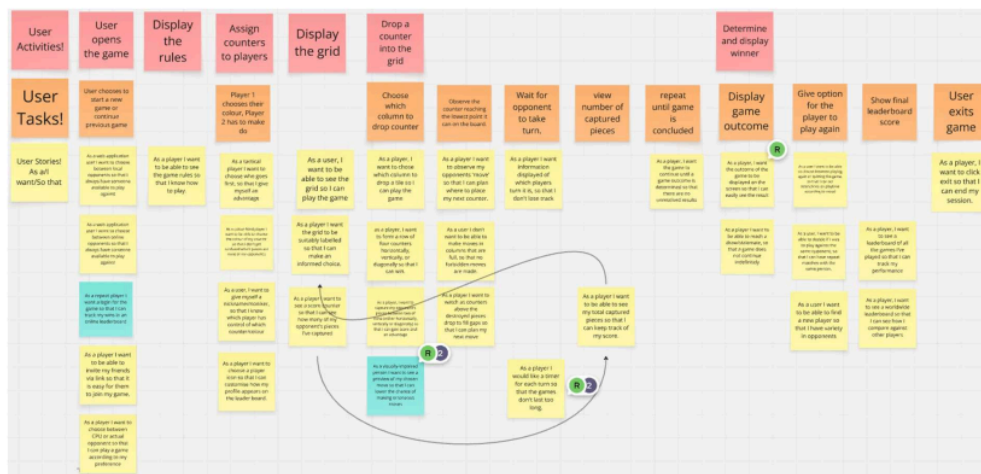


Fig.9: A screenshot showing the User Stories Map created for the Connect 4-style game. Created in Miro.

This map shows a narrative flow of tasks and activities, with the user going from opening the game, to then playing and exiting the game. Pinpointing these tasks and activities on a map shows both what the user will be doing during their gameplay session, in addition to mapping out their potential experience, their journey, with the game. Pinpointing and mapping these tasks also help with identifying the essential components needed to create the game, being a way to identify the fundamental core of the software.

These tasks also allow us to group user stories together in rows, which will correspond to each activity. Each task is placed in a certain position, which was decided by the team in order of importance from top to bottom. This allows the team to decide on what is essential for the game to both function and work for the user's specifications and requirements. Each story is condensed and short, which is a necessity for good user stories.

This is one of the stories on the board, underneath the task of dropping a counter into the grid.

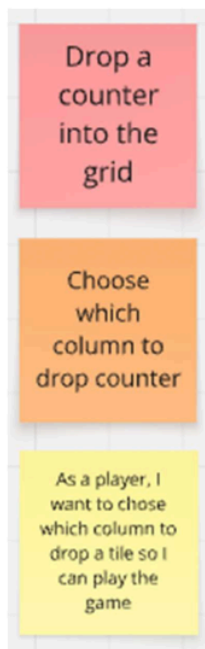


Fig.10: A screenshot showing a section of the User Stories Map created for the Connect 4-style game. Created in Miro.

Looking at the story, the benefit of the story is clearly shown. From a technical perspective, being able to drop a tile means that the software is working correctly without any bugs or issues in the code. However, from a user perspective, the benefit isn't just the code working, but rather the player is allowed to play the game and experience it for himself.

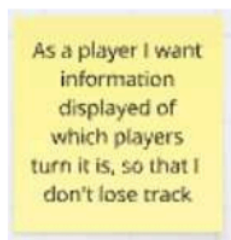


Fig.11: A screenshot showing a section of the User Stories Map created for the Connect 4-style game. Created in Miro.

This user story shows a good example of a story that doesn't have a clearly technical benefit to it. On the technical and coding side, you'd only need to implement a way to change the

turn order, from player 1 to player 2 and vice versa. However, the benefit of this story is that, having that extra visual feedback to signify that it is their turn currently, allows the user to know when they can play and thus improves on their personal experience with the game.

Overall, the team's approach to the user story map was well thought out and structured, outlining many potential specifications and features that can be developed upon for the final product. The user story map also allowed the team to collaborate in terms of ideas and to communicate them across in a clear and understandable manner. It gave the team some acceptance criteria for further design work, such as the CRC cards. However, certain elements that may have been low priority, such as the turn timer user story for example, would be potentially neglected in the final design due to being of low importance.

3: CRC Card Design Approach

Class-Responsibility-Collaboration Cards, known commonly as CRC Cards, are a design tool created by Kent Beck and Ward Cunningham [8] which are commonly used in the planning stages of software development. CRC cards consist of 3 sections: a name for the class, the responsibilities, which are the tasks and functions they perform, and the collaborators, which are the other classes that the class will communicate with in the program.

For the Connect 4-style game, CRC cards were generated with the acceptance criteria from the User Story Map.

William Baker Oliver: Software Engineering Principles and Testing



Fig.12: A screenshot showing a portion of the CRC cards created for the Connect 4-style game. Created with Miro.

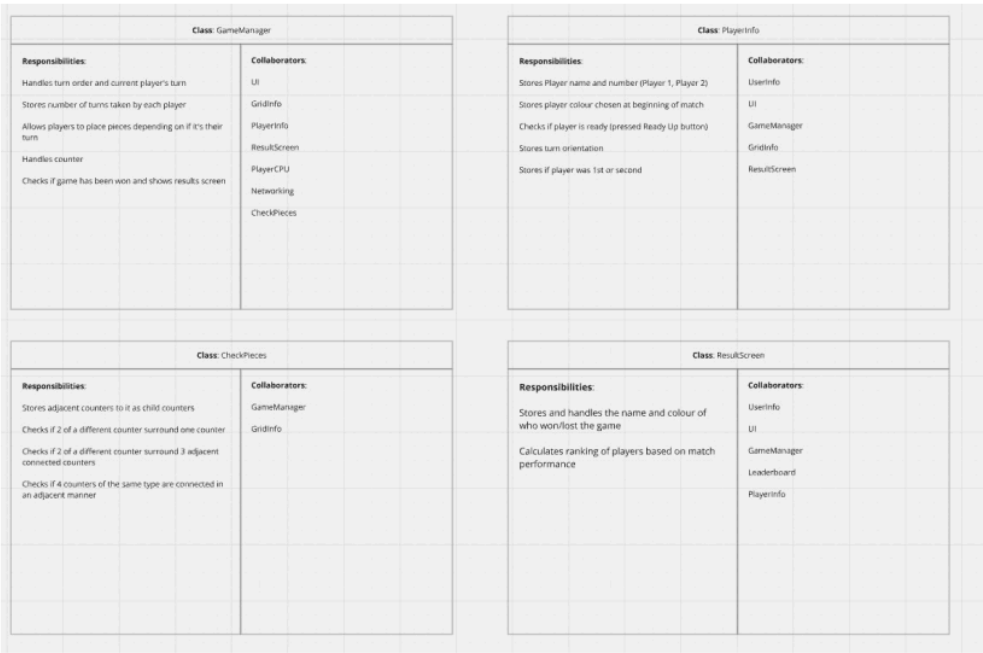


Fig.13: A screenshot showing a portion of the CRC cards created for the Connect 4-style game. Created with Miro.

The cards produced here all adhere to the criteria and potential features outlined in the map. For example, one of the user activities was to display the game's outcome. Since this was sectioned off as its own Class, to handle this important task for the user. The responsibilities in many of these classes also adhere to the map, with many being based on the user stories and the potential requirements for the user.

Class: Leaderboard	
Responsibilities:	Collaborators:
Stores global player data	UserInfo
Allows player to view other player's ranking in the menu	UI
Gets player ranking data from finished matches	Networking
Updates global leaderboard online with player's current rank	ResultScreen

Fig.14: A screenshot showing the Leaderboard CRC card created for the Connect 4-style game. Created with Miro.

Take this Leaderboard class for example. This class corresponds to the section of the user story map that suggests one of the user activities is to show the leaderboard score. The responsibilities line up with the expected function of the class, along with the user stories for a potential leaderboard, where they want to see their scores from finished matches, along with seeing a global leader board, for the benefit of tracking and comparing their performance with other players.

Many of the responsibilities are condensed but have enough detail to explain their use and functionality, such as the Leaderboard functionalities. It's easy to see and understand the requirements for this class, being the storing of data, viewing of match data, and obtaining data from a global leaderboard, all the while working with other classes to achieve this. From these cards, a potential class and functions for the software could be tested and built. Another advantage of these cards is that many functions are separate, such as the UI and ResultScreen.

Overall, these CRC cards aligned very closely with the acceptance criteria outlined in the user stories and the user story map, with many cards containing enough detail for potential development and testing, however certain elements could be expanded upon. For example, one user story states that the user would like to not place a counter in a row which is full. The GridInfo board, which handles placing counters on the grid, doesn't explicitly state that it prevents the player from doing this, only that it manages the placement and ownership of counters on the grid. This could be expanded further to better communication and give greater focus to issues like this in development.

4: Software Testing Evaluation

Software testing is a vital part of any software development process. Paul C Jorgensen states that the two main reasons that software is tested is to "make a judgment about quality or acceptability and to discover problems" [9]. It ensures that the quality of the software is of a high standard, and that it works as intended.

It is vital that software is thoroughly tested throughout development, as releasing or deploying untested software with bugs or other potential issues can greatly impact users. A notable example of this occurred as recently as 2024, where the IT security company CrowdStrike sent out a software update to their customer base. The update turned out to be corrupted, and it caused computers running their software to crash every time they started up [10] [11]. This led to a global outage of nearly 8.5 million devices running Windows [12], causing major issues worldwide and many services, such as airlines for example, which used the CrowdStrike software, to shut down. This caused thousands of cancellations and all services to be stopped at multiple airports until a fix was found [13]. This incident alone shows the severe importance of testing code before deployment. As software develops over time, the issues or bugs they may have increased over time, which will render any development process mute until fixed. It's important to test your software consistently as it develops.

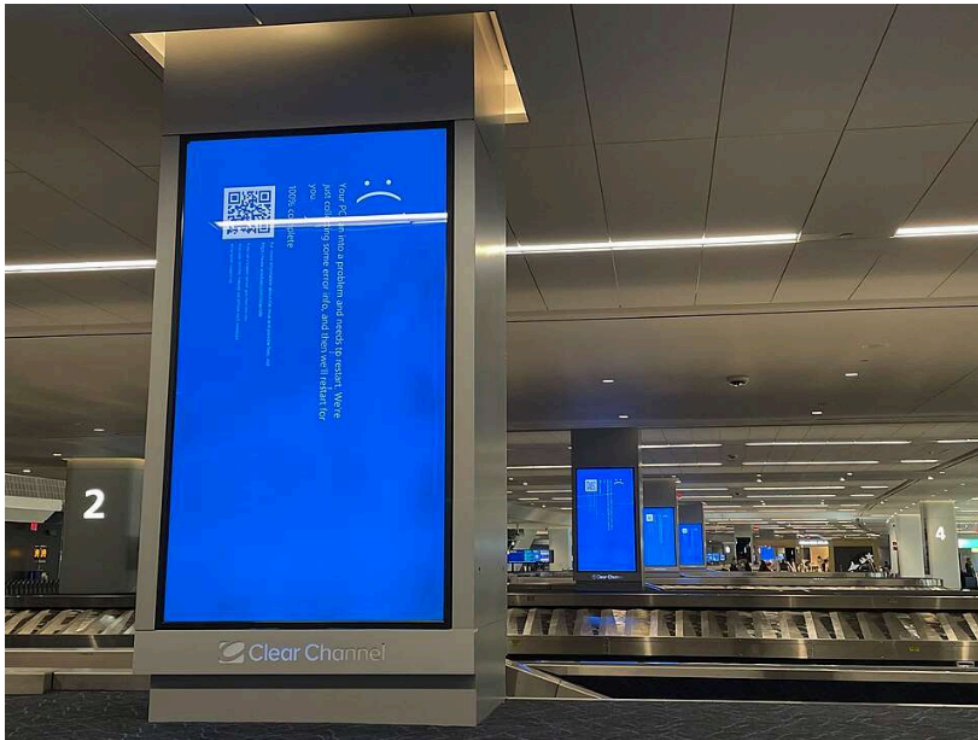


Fig.15: A picture showing baggage carousels at LaGuardia Airport, with the screens in the area displaying a Blue Screen of Death because of the CrowdStrike outage. Taken from [14].

It is also crucial that software testing is done thoroughly. Take the Connect 4-style game for instance. You would want to test the game to ensure that the pieces fall as intended when placed, and that the extra rules such as capturing or destroying pieces are properly integrated. Whilst initial testing may show that capturing a counter with 2 other pieces may work as intended, other conditions need to be tested as well, such as it working for each player's counters, as well as ensuring that counters that drop down from other counters being destroyed will not destroy or capture counters.

With regards to software testing, there are two major approaches that can be used: Black Box and White Box testing. Black box testing, as the name implies, is a testing methodology that does not concern the internal workings of the software at all. Rather, the software's functionality is tested from the outside, with no knowledge of any of its code [15]. Black box testing is very similar to user stories in this aspect, as they focus on the desired specifications and functionality for the user, rather than the technical aspect. This approach is mainly taken by QA and software testers.

When doing black box testing, there are a variety of methods and techniques that can be used. One example is known as Equivalence Partitioning. When software is being tested, you want to ensure that all potential areas are covered. However, take a program which takes a value as an input. It can only take values within a certain range, such as 20 to 65. However, testing for each potential input within the range and outside is time-consuming and not feasible. You would need a way of finding that this range works, without needing to test every single value. This is where Equivalence Partitioning comes in. Instead of testing all potential values, you can create separate them into groups, or a partition, to test, which are known as valid and invalid partitions [16]. So instead of “testing every value less than 20”, you can group it into an invalid group of “less than 20”, and test one singular value which meets the criteria, e.g. 15. Same goes for values within the range and greater than the range, which can be grouped into valid and invalid groups accordingly. By using this method, you can essentially reduce the number of tests from being potentially infinite, to a small finite amount, all whilst covering the potential areas of testing needed.

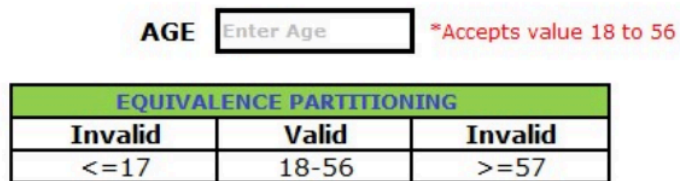


Fig.16: A diagram showing an example of an Equivalence Partitioning table, showing the 3 different groups that can be used to test an age input. Taken from [17].

On the other hand, white box testing is very much the opposite of black box testing. According to The Art of Software Testing, white box testing “permits you to examine the internal structure of the program” [18]. It focuses on examining the code and structure of the software to ensure that everything works and functions on a technical level. As this kind of testing requires extensive knowledge of the code and structure of the software, this testing approach is mainly taken by software developers.

In many instances, it is suggested to use a mix of both black box and white box testing during development. The main issue with white box testing is that it focuses only on the functionality of the code, whilst black box deals with user issues with the software on the outside. Due to this, both testing methods become foils to each other and complement each other as a result: the black box helps focus on the user experience and potential issues the

user faces, whilst while box focuses on debugging the entire code to ensure everything runs smoothly behind the scenes. A combination method exists called Gray Box testing, which essentially acts as a fusion of the two methods, where the player tests the outside functionality of the program, such as “context-specific errors related to web systems”, whilst the tester has partial knowledge of the inner workings of the software [19].

The simplest form of white box testing that exists is known as Statement Coverage. Simply put, it checks and ensures that each line of code executes, and functions as intended [20]. However, another form of white box testing exists, which is known as Path Testing. Path testing, unlike statement coverage which only checks if each line of code executes, ensures that the code flows and executes as intended, going through all potential statements and loops. Think of some code with if and else statements. You would want to ensure that each line within the statements work as intended, but also that the code can flow properly through each possible path, ensuring that it executes the correct statements depending on certain conditions. Many times, a flow chart will be used to illustrate the potential paths the code could take in testing.

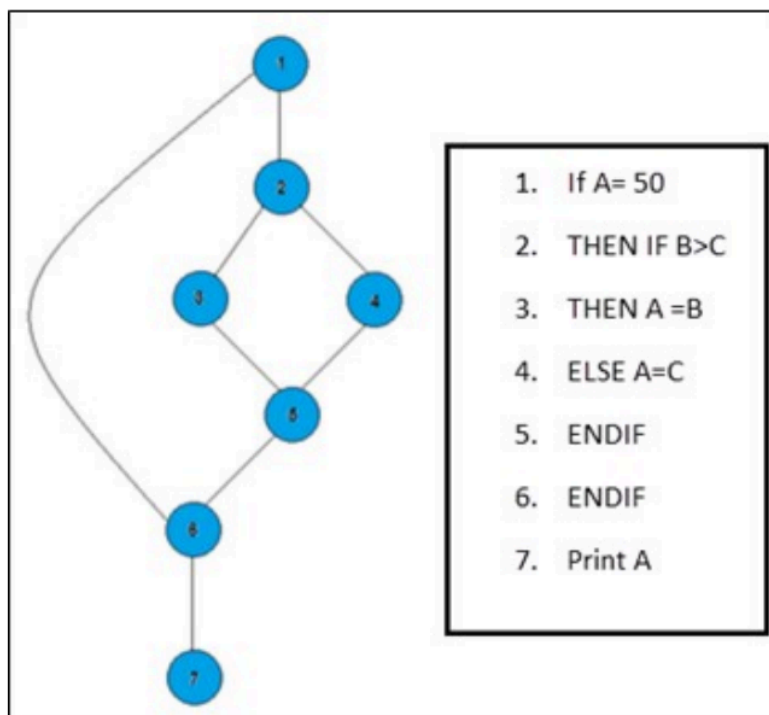


Fig.17: A diagram showing an example flow graph for some example code, showing the multiple paths that the code could take, which would need to be tested for in path testing. Taken from [21].

An important aspect of developing code is ensuring that the code written is feasible enough to test. Mauricio Aniche states that “Design for testability is fundamental if our goal is to achieve systematic testing—if your code is hard to test, you probably won’t test it.” [22]. It is essential that, from the beginning of the software and code being written, that it will be easily testable later. The design and structure of well thought out code revolves around this concept. If code is hard or not designed to be testable, issues may arise, such as testers not being able to find potential bugs in the software or debugging and testing methods being more time consuming as a result, which slows production down. Essentially, coding must be done with testability in mind for a successful deployment.

A common way code is made easily testable is to not create objects that a class is dependent on within a class, rather instead allowing to be injected via a class constructor. This is known as Dependency Injection [23], and doing this allows the tester or programmer to pass through what is known as a mock function. This essentially acts as a simulated object that acts as the legitimate object that needs to be passed through the class, to allow them to test the code to ensure that it is working properly.

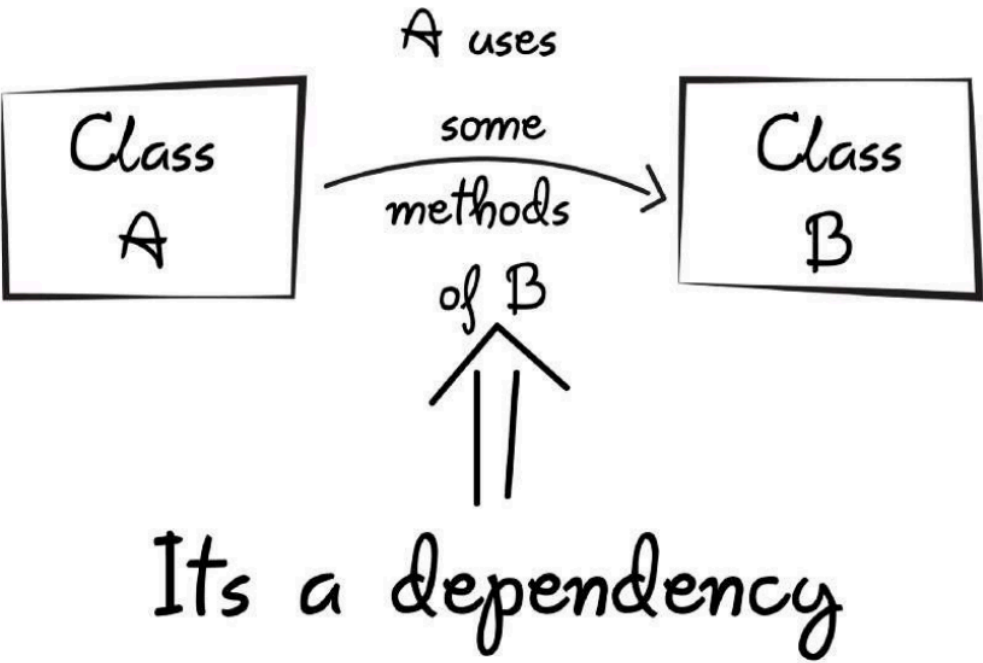


Fig.18: A diagram showing how dependency injection works, showing one class using methods from another class. Taken from [23].

Going back to the Connect 4-style game, there are many tests that can be potentially performed with both black and white box testing methods. Shown below are examples of black box tests that apply to the game.

Pre- Conditions (Setup)	The username input box is blank for a name to be entered and for the player to confirm the name.
Inputs	Invalid: (nothing is entered in the box, intentionally left blank) Valid: Username1234 Invalid: %

Expected Output	The invalid ones should turn up a message on screen stating that the name is invalid, and that the user should enter a different one, whilst the valid name should go through.
Description	This tests for the username input system, ensuring that invalid names such as blank names or names with special characters cannot be entered.

Pre- Conditions (Setup)	The game shows a register screen, with 2 boxes waiting for a username and password to be entered.
Inputs	Username: UsernameTest2 Password: password6
Expected Output	The software should check the input criteria and accept the credentials entered by the user, and store it within the server
Description	This tests for the sign-up system on the menu, to ensure that players can properly sign up with a valid username and password.

Pre- Conditions (Setup)	The 6x7 grid shows 3 counters of the same colour stacked on top of each other, in the first column of squares.
Inputs	The player will drop a counter in the first column.
Expected Output	The counter should drop down to the other 3 counters, where the game will register that the player has won, drawing a line over the counters and displaying a message.
Description	This tests for both the counters falling onto other counters, in addition to testing the win condition for the game.

In addition to black box tests, there are certain white box tests which can be applied to the game's code, which are shown below.

Pre- Conditions (Setup)	GameGrid's GridInfo[6][7] has nothing but empty values.
Inputs	The function CounterDrop(location = 0, player = 1) will be called.
Expected Output	The counter should move down the array, going to value 5 (row 6).

Description	This tests to see if the counter will go to the lowest possible area in the grid, represented by the array.
-------------	---

Pre- Conditions (Setup)	The values at GridInfo[0][5] will be filled with player 1's counters, which are seen as 1 in the array.
Inputs	The function CounterDrop(location = 0, player = 1) will be called.
Expected Output	The code will return a debug message in the console, stating that "Counter cannot be placed, array column is full", and skip the condition: If (columnCheckFull == false)
Description	This tests to see if the code will check the condition for seeing if the array column is full and skip placing a value down in the array as a result.

5: Application and Investigation of Unit Testing

According to Gayathri Mohan, Unit Tests are tests that "aim to create safety nets at the micro level. They validate the smallest portions of an application's functionality; for example, a unit test might verify a method's behavior in a class." [24]. It essentially sections off certain parts of the code, such as methods and functions, and then those sections are tested separately from the rest of the code. It allows for functions to be tested in small sections, which assists with the testing process, making it faster to test due to the small scope and can capture bugs easily as a result [25]. Code must be developed around unit testing as a result, so ensuring that the code is designed to be testable in small sections is a necessity during development. One way this can be done is with a Test Harness. A test harness is a collection of test data and code which is designed to test specific sections of a program, using various inputs and scenarios. Unit testing also can use mock functions to perform tests.

Unit testing can be applied to the white box tests suggested for the Connect 4-style game. Take the test that checks to see if a counter drops down. An empty grid array of Grid [6][7], a player, and a counter colour can be initialised and passed through the function CounterDrop(), where upon the counter reaching the bottom of the array column, a debug or console message can be sent, detailing where the counter ended up at and whether the test passed. Unit testing can also apply to the path test for checking if a column is blocked.

Initialising a grid array with one column full of the same counter type, then using the CounterDrop() function, it can then check its own if statement to see if columnCheckFull is equal to true. If so, it will return a debug message stating that the column is full, and that no more counters can be placed there. Unit testing can assist with white box tests, isolating certain parts of the code to be tested in small sections, along with using other methods such as sending mock functions or by using a test harness integrated into the software.

Conclusion

In conclusion, this assignment demonstrates the utility and importance of designing and testing software, in addition to team communication within the software development process. By using design methodologies such as User Stories and CRC cards, it ensures that teams have a way to collaborate on ideas and can create a focus and objective for both designers and developers. Testing methodologies and approaches such as black box, white box, grey box and unit testing ensure a high-quality standard for software, in addition to helping improve the user experience in the process. These elements all make up a vital component of any software development process and can help allow teams to create high-quality software.

Appendix

External Links

Link to Wireframe Diagrams:

https://miro.com/app/board/uXjVI7VWYVo=/?share_link_id=701416475048

Link to CRC Cards:

https://miro.com/app/board/uXjVI72W524=/?share_link_id=191993451217

Link to UML Design:

https://miro.com/app/board/uXjVI7UutAA=/?share_link_id=873438517571

Bibliography

- [1] Miro, "Miro | The Innovation Workspace," Miro, 15 April 2025. [Online]. Available: <https://miro.com>. [Accessed 15 April 2025].
- [2] M. Fowler and K. Scott, UML Distilled: A Brief Guide to the Standard Object Modeling Language, 3rd ed., Boston: Addison-Wesley Professional, 2003, p. 14.
- [3] Visual Paradigm, "What is Sequence Diagram?," Visual Paradigm, 15 April 2025. [Online]. Available: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/>. [Accessed 15 April 2025].
- [4] P. Guilizzoni, "What are wireframes and why are they used?," Balsamiq, 15 April 2025. [Online]. Available: <https://balsamiq.com/learn/articles/what-are-wireframes/>. [Accessed 15 April 2025].
- [5] M. Cohn, User Stories Applied: For Agile Software Development, London: Pearson Education, 2004, pp. 4-5.
- [6] M. Cohn, "User Stories: How to Create Story Maps," Mountain Goat Software, 1 October 2024. [Online]. Available: <https://www.mountaingoatsoftware.com/blog/user-story-mapping-how-to-create-story-maps>. [Accessed 15 April 2025].
- [7] J. Patton, User Story Mapping, Sebastopol: O'Reilly Media, 2014, p. 21.
- [8] K. Beck and W. Cunningham, "A Laboratory for Teaching Object Oriented Thinking," *SIGPLAN notices*, vol. 24, no. 10, pp. 1-6, 1989.
- [9] P. C. Jorgensen, Software Testing: A Craftsman's Approach, 4th ed., Boca Raton: CRC Press, 2014, p. 3.
- [10] J. Tidy, "CrowdStrike IT outage affected 8.5 million Windows devices, Microsoft says," BBC News, 20 July 2024. [Online]. Available: <https://www.bbc.co.uk/news/articles/cpe3zgznwjno>. [Accessed 15 April 2025].
- [11] N. Robins-Early, "What is CrowdStrike, and how did it cause a global Windows outage?," The Guardian News & Media, 19 July 2024. [Online]. Available: <https://www.theguardian.com/technology/article/2024/jul/19/what-is-crowdstrike-microsoft-windows-outage>. [Accessed 15 April 2025].
- [12] D. Weston, "Helping our customers through the CrowdStrike outage," Microsoft Corporation, 20 July 2024. [Online]. Available:

<https://blogs.microsoft.com/blog/2024/07/20/helping-our-customers-through-the-crowdstrike-outage/>. [Accessed 15 April 2025].

[13 A. Marshall, "Why the Global CrowdStrike Outage Hit Airports So Hard," Wired, 19 July 2024. [Online]. Available: <https://www.wired.com/story/crowdstrike-windows-outage-airport-travel-delays/>. [Accessed 15 April 2025].

[14 Wikipedia, Wikipedia, 15 April 2025. [Online]. Available: https://en.wikipedia.org/wiki/2024_CrowdStrike-related_IT_outages. [Accessed 15 April 2025].

[15 GeeksForGeeks, "Black Box Testing – Software Engineering," GeeksForGeeks, 9 April 2025. [Online]. Available: <https://www.geeksforgeeks.org/software-engineering-black-box-testing/>. [Accessed 15 April 2025].

[16 TutorialsPoint, "Software Testing - Equivalence Partitioning Testing," TutorialsPoint, 15 April 2025. [Online]. Available: https://www.tutorialspoint.com/software_testing_dictionary/equivalence_partitioning_testing.htm. [Accessed 15 April 2025].

[17 Rajkumar, "Equivalence Partitioning Test Case Design Technique," Software Testing Material, 13 February 2025. [Online]. Available: <https://www.softwaretestingmaterial.com/equivalence-partitioning-testing-technique/>. [Accessed 15 April 2025].

[18 G. J. Myers, T. Badgett and C. Sandler, The Art of Software Testing, 3rd ed., New Jersey: John Wiley & Sons, 2012, p. 10.

[19 GeeksForGeeks, "Gray Box Testing – Software Testing," GeeksForGeeks, 3 March 2025. [Online]. Available: <https://www.geeksforgeeks.org/gray-box-testing-software-testing/>. [Accessed 15 April 2025].

[20 GeeksForGeeks, "Statement Coverage Testing," GeeksForGeeks, 6 January 2025. [Online]. Available: <https://www.geeksforgeeks.org/statement-coverage-testing/>. [Accessed 15 April 2025].

[21 T. Hamilton, "Path Testing & Basis Path Testing in Software (Examples)," Guru99, 4 April 2024. [Online]. Available: <https://www.guru99.com/basis-path-testing.html>. [Accessed 15 April 2025].

[22 M. Aniche, Effective Software Testing: A Developer's Guide, Shelter Island: Manning Publications Co., 2022, p. 173.

- [23 B. Karia, "A quick intro to Dependency Injection: what it is, and when to use it,"
] freeCodeCamp, 18 October 2018. [Online]. Available:
<https://www.freecodecamp.org/news/a-quick-intro-to-dependency-injection-what-it-is-and-when-to-use-it-7578c84fa88f/>. [Accessed 15 April 2025].
- [24 G. Mohan, Full Stack Testing, Sebastopol: O'Reilly Media, 2022, p. 52.
]
- [25 Amazon Web Services, "What is Unit Testing?," Amazon Web Services, 15 April 2025.
] [Online]. Available: <https://aws.amazon.com/what-is/unit-testing/>. [Accessed 15 April 2025].

Assignment

GRADEMARK REPORT

FINAL GRADE

73 / 100

GENERAL COMMENTS

Eval Des (20): 13

User Stories (20): 14

CRC (20): 14

SW Testing (20): 18

Unit Testing (20): 14

FEEDBACK:

=====

It was interesting that you started with a discussion of UML and I liked the wireframe diagrams.

I would have liked a little more discussion of the stories and map that you helped produce. Likewise the CRC cards.

There was some evaluation of the process but was a bit brief, I was expecting more discussion of the benefits and drawbacks.

Liked the discussion of SW testing. Although the unit testing evaluation could have given some example or technical description.

SW tests for the app are described.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25
