

Software Engineering Design & Testing James Frew 2108660

by James Frew

Submission date: 28-Apr-2023 06:01PM (UTC+0100)

Submission ID: 203928891

File name:

31653_James_Frew_Software_Engineering_Design__Testing_James_Frew_2108660_490085_587193704.docx
(745.92K)

Word count: 6802

Character count: 32575

Software Engineering Design & Testing

James Frew
Student University of Wales Trinity St
David
Swansea, Wales
2108660@student.uwtsd.ac.uk

Abstract—Software testing is becoming a larger and larger part of the software development process. Its design methods of User Stories, CRC Cards, Wireframes, UML, Flow Diagrams, and Sequence Diagrams all work to ensure that the likelihood of creating an error is lessened. Finding an error in the project design phase drastically reduces the time and expenses required to fix it, rather than discovering the error at the end of the project. Errors are still likely to occur, so software testing elements of White box and Black box testing help discover them from the user, and software points of view (respectively). A Connect Four game is designed using the above methods then its potential black, white, and unit tests are examined.

Keywords—User Stories, CRC Cards, Black Box, White Box, Unit Testing

I. INTRODUCTION

This report was created to examine the methods that go into designing and producing a piece of software. This report examines the concepts of User Stories, CRC Cards, Wireframes, UML, Flow Diagrams, and Sequence Diagrams that have been created to produce a game of Connect Four (with additional rule amendments). The report then focuses on how the software would be tested after it's been made – when and how Black box and White box testing would be used. It explores the individual methods that each test uses, with extra detail assigned to the research of unit testing, and how it could be applied to the designed game.

II. PROJECT DESIGN

A. User Stories

The design produced for the user stories covers most of the aspects of the game. The requirements of playing the game are almost all present – with enough to cover the MVP. With the current user stories, the user would be able to play a game against another player and achieve a victory against them. Additional stories that notify the user of a loss/draw should be added, along with the ability for user account creation. Currently, a system would have to be introduced to allow users to make an account with which to play with. Otherwise, there would be no way of playing the game.

The primary goal that was decided upon by the group was to “Allow two users to play connect four on a web application”. This goal is covered by the user stories and has potentially been overdone with the introduction of user accounts and online play – as there is no listed requirement that the users would be playing remotely – but the group decided that this would be more beneficial enough to include.

The group worked effectively when producing the user stories. A great deal of discussion when into how much detail should be included in each story, and what information should be separated into new ones – though this would be the main point of discourse for the group at this stage.

Eventually the user's score, as well as the two additional rules were placed into the second release phase. The score was deemed as a non-essential aspect of the Connect Four game, so could be left until later. Similarly, the additional rules were listed as amendments, so although essential to the overall criteria, these could be left until later if needed. The group agreed that the game would still be recognizable as Connect Four if these features were postponed. The whole group worked as a team without one individual dominating the conversation, and each member was able to get their views across.

One of the core user stories identified by the group was the one in which the user was able to find an opponent (“As a user I want to be able to find an opponent to play against so that I can play the game online”). This story would be essential to any user who wanted to play this *multiplayer* game. Another essential user story was the ability to place counters in the grid (“As a user I want to be able to select which column I want to drop my counter into so that I can play the game”), though not nearly as beneficial as the story which precedes it – being able to see the grid to place the counters (“As a user I want to be able to see the grid that the game is played on so that I know where I can place counters”). This is a necessity to be able to play, as users wouldn't know where counters were or were not currently placed.

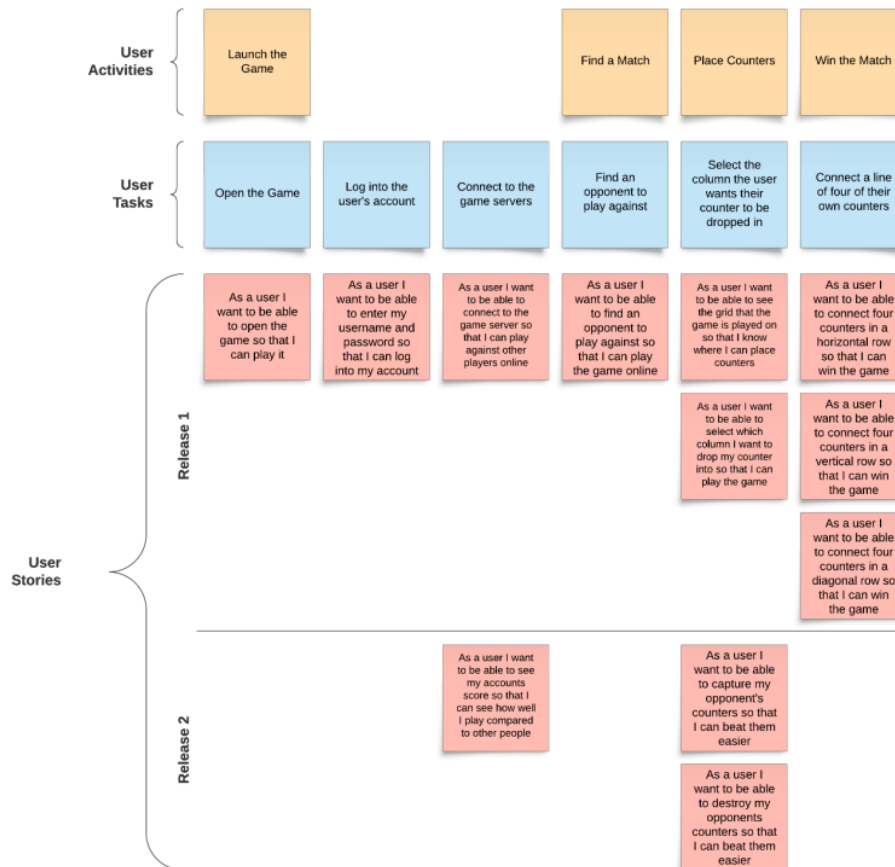
The three final stories which involved the different methods of winning the game really benefit the user in the variance in which they are able to win (“As a user I want to be able to connect four counters in a horizontal row so that I can win the game”, “As a user I want to be able to connect four counters in a vertical row so that I can win the game”, “As a user I want to be able to connect four counters in a diagonal row so that I can win the game”). Having only one of these stories would make the game feel very different, be much harder to win (resulting in more draws) and therefore a lot less entertaining to play.

The group became a lot more divided when it came to allocating story points. The general trend for scores was to have one score on each extreme and a general grouping somewhere between the two. This created a lot of discussion as to why the two outlier members had such a strong inclination towards their chosen difficulty. Most commonly,

the reason for these two results were due to a difference in individual knowledge or skills. This was most prevalent in the user story which involved connecting to the game servers (“As a user I want to be able to connect to the game server so that I can play against other players online”). Students studying game development rated this very highly – having no idea where they would start tackling the problem of servers. In contrast, those who had studied

networking, databases, or both, more fully understood the area and rated it as easier. Discussion of this led the Games development students to lower their scores after being informed on the actual complexity of the task.

This need for different knowledge/skill bases arose quite frequently, which allowed each member to explain the difficulty of their field to the others.



B. CRC Cards

The group examined the user stories in search of their nouns, discarding several duplicates, attributes, and vague/irrelevant nouns. Lots of group discussion was had generating the cards, eventually settling on three – “User”, “Game”, and “Turn”.

The “User” CRC card would provide all the information about each user’s account – password, username, email, etc. This information would be used to allow them to connect to the game servers, accessing the game and viewing their scores. The group determined that the only other card that this CRC card would need to interact with would be the “Game” card – as the “Turn” card wouldn’t require any account information to function.

The “Game” card was decided as the bridge between users and the turns they would take. This CRC would

therefore collaborate with both the “User” and “Turn” cards. It would also manage the connection between users and display the grid for them to play on. It also calculates the change in user’s scores and updates the information to the user’s accounts. There was a lot of debate over this decision as some members of the group believed that the score should be placed in its own CRC card, but it was eventually concluded that it would stay in the “Game” card. In hindsight, placing it separately would also have worked more neatly as it would function fine on its own.

The “Turn” CRC card was designed to house the actions that users would take on their turns. It handles allowing a user to place a counter, as well as the additional rules of capturing and destroying opponent’s counters. The group decided to section these aspects off here to allow easy repetition – as would be frequent each game. The only

collaboration needed is with the “Game” card as the user’s accounts are not needed for the counter interactions.

However, the group did miss a few parts from the acceptance criteria from the user stories. There is no responsibility listed for displaying if a user has lost or drawn a game – only one for if they’ve won. Additionally, the group overlooked the need for assigning the users to either player 1 or player 2. Both would come under the “Game” CRC card.

Overall, the group was able to identify most of the elements of each CRC card and organize them accordingly.

User	
• Provide user account information. • Allow users to connect to the game server. • Store users scores.	• Game

Game	
• Allows users to play against each other. • Provide the grid to show where counters can be / are currently placed. • Determine if a player has won. • Update user score.	• User • Turn

Turn	
• Allow a user to place a counter. • Allow a user to capture an opponent’s counters. • Allow a user to destroy an opponent’s counters.	• Game

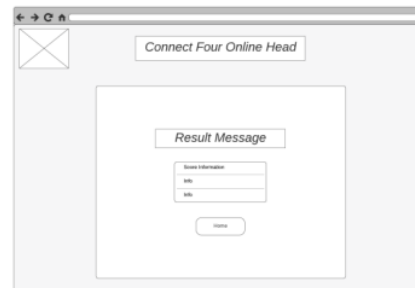
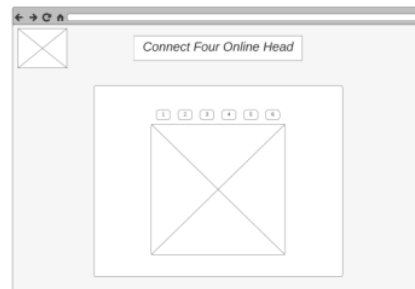
C. Wireframes

The group design of the wireframes for the website was completed relatively quickly using a minimalistic website layout. Lucid was used to create the designs – owing to its many useful templates.

The website features a header of “Connect Four Online” with a box below it in which the game would be played. The top left image would be the site logo. The first picture shows the login page, with two text fields and a login button.

The second picture shows how the game would appear when played. The 6 boxes above the image represent buttons to be pressed to decide on which column to drop the player’s counters. The image would function as a visual display of the grid, with animations showing the counters dropping down into position.

The final picture shows the result screen. The result of the game – win, lose, or draw – would be displayed to each player after the match. Underneath it would be the information about the player’s score changes – previous score and updated score.



D. UML

A UML diagram could be an alternative, or assisting diagram to the CRC cards and would be derived similarly. The main benefit of UML is that it shows potential attributes, operations, and associations between different classes.

The “User” class contains the attributes to store the user’s username, password, and email. These are all strings and private. The group decided this class would contain the operations for logging in, and connecting the user to the game servers.

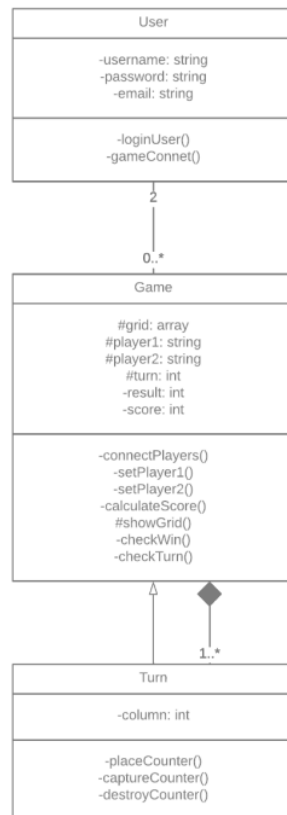
The association between the “User” class and the “Game” class would be 2 users to 0+ games – as one game can only have two players, but there is also the possibility that no players will have any games (as it’s possible to have an account and not play).

It was decided that the “Game” class would contain attributes for the grid, player 1, player 2, the current turn, the game result, and the score. The group missed out the current player attribute when designing the UML, and instead used a turn counter. The idea was that this would track which turn number it was, which would be used to decide whose go it was. This would not be the final chosen method as the flow diagram would help reveal a better

alternative. The operations present would handle connecting the players, setting the users to each player position, calculating the score, displaying the grid, checking for wins, and checking whose turn it is.

The association between “Game” and “Turn” is that “Turn” would inherit the protected attributes of grid, player1, player2, and turn, as well as the display grid operation. The “Turn” class has a composition requirement on the “Game” class – as turns cannot exist without a game existing. At least one turn is needed in each game.

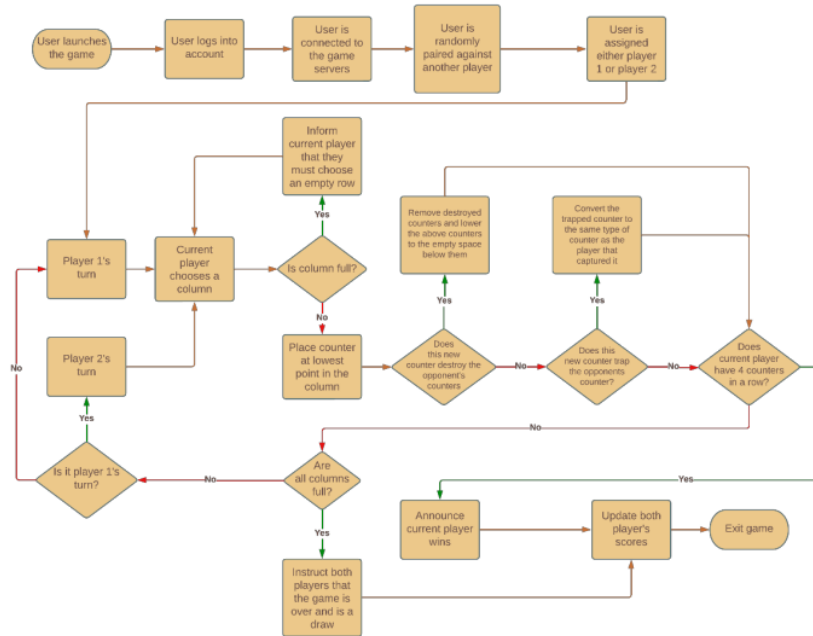
The “Turn” class would house the attribute players would use to decide which column to drop the counter in. It would also have the operations to place, capture, and destroy counters.



E. Flow Diagram

The group designed the flow diagram to demonstrate all the paths, choices, and outcomes that users would take whilst using the program. Its first path walks through the user launching the game, to player 1 starting their turn. Following this, the diagram illustrates the conditional statements that the user will enact – checking for full columns, destroying, capturing, winning, and checking if all spaces are full. The flow diagram shows how the program will loop (alternating players) until either all the spaces are full, or a player has won the game. It then instructs the players on the outcome of the game and exits itself after calculating and showing the scores.

The flow diagram was likely the most useful diagram in determining potential black and white box tests – it really helped to visualize the process that was taking place. The group agreed that they would create this before the UML in future.



F. Sequence Diagram

The group found it more of a struggle to design the sequence diagram. This was believed to be due to how large of an area the diagram took up. The overall layout of the diagram is ok but starts to get slightly confusing towards the

end. Visually, the diagram would flow a lot better if the victoryCheck and victoryScreen came after the fullCheck and gridFull. The chosen design originated from following a variation of the happy path, in which the result happened before the grid became full.

everything to work as is expected and the Happy Path [4] (best route) to be followed. Testing to fail should be performed after testing to pass, and concerns values that *should* cause the system to fail. This allows for testing any system weaknesses/vulnerabilities, or other unexpected problems or failures. During this, boundary conditions and incorrect inputs should be entered as much as possible to try and accomplish exhaustive input testing [5].

Unfortunately, full exhaustive testing is not viable as the time and cost requirements would exceed the value it would bring to the project. This is why test cases are broken up into more effective sets of tests with specifics being testing – boundary conditions, combinatorial testing, etc. [6]

B. White Box Testing

White box testing is driven by examining a programs logic and must come after black box testing. Code is examined by visually inspecting it and by running it to find errors. Exhaustive Path testing involves testing every route through a program, function, class, etc. to determine what works and what doesn't. Time and costs limit how fully the program can be tested [7]. Path testing won't catch data-sensitive errors as there are too many possible values to examine. Debuggers are often used here to allow the placement of break points to follow the data as it goes along.

When first encountering an error, the tester must ensure they can first reproduce it, to ensure they know specifically what error they've discovered. After this, they must narrow down the section of code to find a demo of the defect, then write an automatic test which will fail until the error has been corrected. Only then is the defect fixed.

An automated test framework – or test harness – can also be used to help debug a system. This is a collection of software and test data that automatically inputs values and receives outputs to carry out tests on the program. It records instances of successes and failures with each test performed to help speed up testing times [8].

There are several key terminologies that software testing involves: A testing double covers four different types of fabricated objects that are used to stand in for other, real parts of code.

Dummy objects act as stand-in items used to fill parameter lists. Stubs act similarly but have a memorable state to allow them to know what happened in previous iterations they were called – while doing nothing themselves. Fake objects are fake chunks of code that are only there to make the rest of the program function correctly – they won't ship with the final program as they only work as shortcuts to carry out the rest of the program (e.g., using an array in place of an actual database). Mock objects check the behavior of code to identify if the right calls are being made by prescribing their expected results.

An example for demonstrating black and white box testing would be that of a Smart Television system. The black box testing portion would involve pressing buttons on the remote, testing each different outcome from the button input – Functional testing. This would also involve trying to overload the TV's processing capabilities by quickly swapping between different channels – Non-functional testing. Regarding white box testing, the TV's

code would be examined to try and check as many possible paths as possible while using testing doubles to find potential errors.

C. Testable code

The term 'Testable code' refers to the idea that code is written in such a way that it is as easy to test as is feasible with the complexity of the program. Put simply, being able to test the code you've written should not be hard to do when code is written correctly. Testable code is determined from the point of white box testing, not black box. If the way code is written means it cannot be tested using white box, then it cannot be considered testable. Code should be split up, so each section focuses on giving a clear output of its results. This allows unit testing to be carried out much easier and thus speeds up the rate at which code can be tested. Avoiding global variables and ensuring loose coupling helps to achieve this.

Testable code makes fixing errors much easier, so it is a very important aspect of software engineering. Being able to fix errors easily – regardless of what stage of the project cycle you are in – can save a great deal of time and money. [9]

IV. TESTS

A. Black Box

Num	Pre-Conditions	Inputs	Expected Outputs	Purpose
1	Page is open in browser	Press launch game button	Game launches	Tests if the user can start up the game so they can play
2	Page is open in browser	Press launch game button multiple times	One instance of the game launches	Ensures the user can only launch one instance of the game and prevents overloading the servers
3	Game is launched	Press login button	Page to enter login details is shown	User is allowed to log into their account so that they can find an opponent
4	Game is launched	Press login button multiple times	Only one instance of the page to enter login details is shown	Ensures the user can only launch one instance of the login page and does not log into the same account multiple times or overload the servers.
5	Game is launched	Press login button in multiple different tabs	One instance of the game launches in each new tab	Allows multiple instances of the game to load
6	Login Page	Enter correct login details	Login details are accepted, and the user has access to their account	User must be able to log into their account when they enter their details correctly
7	Login Page	Enter incorrect login details	Login details are rejected, and the user is asked to re-enter their details	User must not be able to log into their account when they enter their details incorrectly
8	Login Page	Enter database breaching inputs (SQL injections, etc.)	Login details are rejected, and the user is asked to re-enter their details	Users must not be allowed to attempt database breaches.
9	Login Page open on multiple tabs at the same time	Enter the same login details on each page	Login only allowed on the first instance in which the login request is made	Users must not be able to have multiple instances of their accounts open at the same time. This could prevent matching against themselves for a match which could cause major errors when turns/scores are calculated.
10	Logged in and searching for an opponent	-	Paired against a random opponent regardless of the number of times the test is repeated	Opponents must be randomly assigned to each other, so players are unable to play against friends that deliberately throw games to boost their score. Also matching against an opponent is required for the game to be played.
11	Opponent found	-	Assigned randomly as either player 1 or player 2 regardless of the number of times the test is repeated	There should be no preference in which player goes first or second. If one player is repeatedly allowed to go first their chances of win go up substantially.
12	Current player's turn	Place counters in each different column using their buttons	Counter is placed in column if column is not yet full	User's must be able to place counters on their turn to play the game.
13	Current player's turn	Place multiple counters in different columns at the same time using their buttons	Only the first counter is placed	Users should not be allowed to place multiple counters in the same turn as this is cheating.

14	Current player's turn	Place multiple counters in the same column on the same turn using the same button	Only one counter is placed	Users should not be allowed to place multiple counters in the same turn as this is cheating.
15	Current player's turn	Place a counter in a column that is full	Message informing user to try a different column	Full columns no don't have any space to house new counters. They therefore cannot have new counters placed in them.
16	Current player's turn	Destroy opponent's counters by capturing three of them at the same time	Counters are destroyed	One of the additional rules is that opponent's counters can be destroyed by capturing three of them at the same time. So, this must work
17	Current player's turn	Destroy own counters by capturing three of them at the same time	Players cannot destroy their own counters, so nothing happens	Players should not be able to destroy their own counters, or they will never be able to connect four counters together
18	Current player's turn	Capture opponent's counter/s by trapping them between your counters	Captured counters are changed into the current player's counters	The other additional rule is that opponent's counters can be captured by trapping them between your counters. So, this must work
19	Current player's turn	Capture own counters by trapping them between 2 of your own	Players cannot capture their own counters, so nothing happens	Players should not be able to capture their own counters, as they will win much more quickly than is intended.
20	Current player's turn	Connect four horizontal counters	Game is won and message is displayed	User must be able to win the game by connecting four counters in a horizontal line.
21	Current player's turn	Connect four vertical counters	Game is won and message is displayed	User must be able to win the game by connecting four counters in a vertical line.
22	Current player's turn	Connect four diagonal counters	Game is won and message is displayed	User must be able to win the game by connecting four counters in a diagonal line.
23	Grid is full of counters	-	Game is a draw and message is displayed	If the grid becomes full, neither user will be able to place anymore counters. This would result in an infinite loop of one player trying to place a counter.
24	Opponent's turn	Place counters in each different column while not the player's turn	No action should take place	Users should not be able to place counters when it is not their turn.
25	Opponent's turn	Opponent wins the game	Game is lost and message is displayed	Users must be notified if their opponent beats them, so they know they have lost. Also verifies that users can lose.
26	Game is a win	-	Score increases	Score must increase after a user has won a game.
27	Game is a loss	-	Score decreases	Score must decrease after a user has lost a game.
28	Game is a draw	-	Score stays the same	Score must stay the same after a user has drawn a game.
29	Either player's turn	Game is closed before it finishes	Player who closes the game early counts as a loss and the other player is rewarded with the win screen	Abandoning the game should cause a penalty for the quitter and other player must be made aware of what has happened so that they are not stuck forever waiting for their opponent to take their turn.

B. White Box

Num	Pre-Conditions	Inputs	Expected Outputs	Purpose
1	Player 1 variable	Check default value is NULL	Value is NULL	Ensures that there is no player 1 user associated before a user has joined
2	Player 2 variable	Check default value is NULL	Value is NULL	Ensures that there is no player 2 user associated before a user has joined
3	Current player variable	Check default value is 1	Value is 1	Needs to be set to 1 so that the game starts with player 1
4	Result variable	Check default value is 0	Value is 0	Placeholder value, also ensures that the game results in a draw if a place disconnects before a result is concluded
5	Browser page is open	Program runs when launch button is pressed	Login stage is opened and waits for inputs	Required for the game to be playable.
6	Login stage	Username and password fields have the correct data entered into them	Program checks the details from the database match the details entered, then connects the user to the game server	Users must be able to log into their accounts when they enter the correct username and password for the account
7	Login stage	Username and password fields have the incorrect data entered into them	Program checks the details from the database match the details entered, then refuses the inputs and asks the user to input the correct details	Users must be prevented from accessing any account with which they don't know both the username and password for
8	User successfully logged in	Code to connect to the game servers executes	User is now connected and able to be matched against other players online	User must be connected to the game servers to connect and play with other players
9	Connected to game servers	Program searches for two unpaired users and connections them together in a game	Users becomes connected to the same instance of the game	Two users must be paired together so that they can play against each other. The random aspect is to prevent match fixing
10	Users paired together	One player is randomly assigned the player 1 variable, and the other to the player 2 variable	The values of the player 1 and player 2 variables will be the same as the user accounts that are now assigned to each of them	Randomly assigning a player to the player 1 position is needed to determine who will go first. Assigning it randomly keeps it fair
11	Player 1 and Player 2 variables have correctly assigned values	"Turn" class starts	"Turn" class in running	The turn class is where all the player interactions take place, so must be working after both players are setup
12	"Turn" class has been started	Current player is asked to choose a column to place their counter in	If all the rows in the grid array for that column are 0s, the current player is instructed to choose another column and program loops back to the start of the turn class.	Users must be able to place counters in columns to be able to play the game.

			If they are not, The lowest open row in the grid in the column the player selected will have its value changed from 0 to the value of the current player variable.	They are not allowed to place counters in columns that are already full
13	Counter has been placed successfully	The function to determine if an opponent's counters can be destroyed is run	If this new counter means that three or more of the opponent's counters are surrounded horizontally, vertically, or diagonally by the current player's counters, they are replaced with 0s at their positions in the grid array. The values in the rows above these new 0s are shifted down to replace the new "empty" spaces. The values at the top row of these columns are set to 0s	Users must be able to destroy opponent's counters if they capture three at the same time – as per the rule amendments
14	The placed counter does not destroy any counters	The function to determine if an opponent's counters can be captured is run	If this new counter means that one or two of the opponent's counters are surrounded horizontally, vertically, or diagonally by the current player's counters, they are replaced with the value of the current player variable	Users must be able to capture opponent's counters if they trap one of their opponent's counters between two of their own – as per the rule amendments
15	Grid has been checked to see if any counters are captured	Check the grid array to identify if there are four counters of the value of the current player variable adjacent in a horizontal, vertical, or diagonal row from each other	If there are, set the result variable to the value of the current player and break out of the turn loop	Users must be able to win the game by connecting four of their own counters in a row. This shows if they can do so and breaks the turn loop so that victory can be announced
16	Four counters not connected	Check all the values in the grid to determine if any are equal to 0	If there are no 0s remaining, break out of the turn loop	If all the positions in the grid have been filled by player counters, there is no longer any position that players can place counters into. This results in a draw so much break the loop to declare such
17	Grid has been checked and game is not a draw	If the current player value is 1, set it to 2. Otherwise set it to 1	Current player value should change to either 1 or 2 (different from previous value) and loop should return to the start of the "Turn" class.	This swaps which players turn it is now, based on who has just gone
18	Loop has been existed	If result variable value is 1 player 1 has won. If result variable value is 2 player 2 has won. If result variable value is 0 the game is a draw	If player 1 has won, inform them that they have won and set score increase variable to 1, and inform player 2 that they have lost. If player 2 has won, inform them that they have won and set score increase variable to 2, and inform player 1 that they have lost. If the game is a draw, inform both players of this.	If a win, or draw has occurred, the correct player must be informed of their corresponding result – loser with a loss, winner with a win. A draw would be informed to both
19	Result variable has been checked	If score increase variable is 1, player 1's score increases and player 2's score decreases. If score increase variable is 2, player 2's score increases and player 1's score decreases. If score increase variable is 0, both player's scores remain the same	If player 1 wins, player 1's score should be higher than previously and player 2's should be lower. If player 2 wins, player 2's score should be higher than previously and player 1's should be lower. Neither score should change if the score increase variable is 0.	When a player wins, their score increases, and when they lose, their score decreases. A draw results in no one losing points

20	Score increase variable has been checked	Database results where the user's information is stored are updated to reflect the score changes.	Database values should correspond to the score changes that have taken place.	The change in score must be relayed back to the database so that the user account score can be used and updated for future games
21	Database values have updated	Users are disconnected from the game servers	Users are no longer connected to the game servers	When users are done playing, they should be disconnected from game server so that they can find a new match

V. UNIT TESTING

Unit testing is when a small piece of code is built to test specific aspects of the main code and should be carried out after black box testing (to avoid knowledge of the code influencing testing). This code will help test the features that white box testing is designed to address – variables, boundaries, sub-boundaries, equations, paths, etc. It uses four main phases to fully carry out its function – though these can vary slightly depending on the current program's needs:

- Setup
- Exercise
- Verify
- Teardown

The setup phase is where every variable that will be needed is declared, objects are created, test doubles are established, and anything else that will be needed for testing the code is made. After this step comes the exercise phase, during which the code is run. This allows the tests to be carried out to check for performance, function, etc. Next, in the verify stage, all the results of the tests are gathered and evaluated so that solutions can be created to fix the errors found. Finally, the teardown phase involves shutting down and removing all the parts created in the setup phase – test doubles, variables, objects, etc. are all deleted from the program. This step is the most important as any leftovers could cause problems later down the line [10].

It is important to make sure that good unit testing is carried out. All tests created should be designed in such a way that they cannot introduce new bugs into the system – whilst also carrying out the tests as quickly as possible. Unit tests must be easy to read and execute and only ever test one specific thing at a time. Regardless of how many times they are used, they should always produce the same results if the same input is given. Ideally, good unit tests would even function if the code base changed [11].

A. Application to White Box Tests

Unit testing can be used to perform many of the white box tests identified for the Connect Four game. Variables such as the user details – username, password, email – can all be automated with dummy results, whilst the database that holds the user details can use a fake as an array – rather than a fully implemented database. A unit test would be supplied with a correct set of user details as well as a fake set to test. It can then be set to try a combination of inputs to see if it can cause a bug. They would be the following:

- Correct username and password
- Incorrect username and password
- Correct username and incorrect password
- Incorrect username and correct password

This unit test would try these inputs against the array (fake database) to try and log in successfully. Only the first option should result in a successful login.

Another test would be to check where the user tries to enter counters into a full column. A script would be written to, when given the choice, only place counters into the first column. After the seventh placement, the user should be prevented from placing anymore counters there. The outputs would be stored and if they match up, the test is a success.

This should be performed for all six columns in the grid individually.

A very similar test – that could build off the previous test's code – would be to test if a draw takes place when every space in the grid is full. The test would automate the placement of seven counters in each column, (forty-two counters total) then record the next result it gets – which should be the game telling the users that no more counters can be placed and that the result is a draw. The automatic filling would have to be slightly more varied than alternating each placement bottom to top, left to right, as this would eventually result in a diagonal Connect Four victory for player 1.

VI. CONCLUSION

The methods of designing a software project, and the application of those methods to the Connect Four program, have been examined and proven useful. Alongside this, the methods used to test software have been researched and their necessity explained. Black and white box testing play large roles in the software testing industry and cover both the human interaction and logical progression of the program written. Unit testing – as a specific aspect of white box testing – has been analyzed to show how useful its automatic bug finding features can be.

VII. REFERENCES

- [1] J. C. Westland, "The cost of errors in software development: evidence from industry," *Journal of Systems and Software*, vol. 62, (1), pp. 1, 2002. . DOI: 10.1016/s0164-1212(01)00130-3.
- [2] Anonymous (-08-20T15:04:15+00:00). *Software Testing Accounts to What Percent of Software Development Costs?*. Available: <https://intersog.com/blog/software-testing-percent-of-software-development-costs/> [Accessed Apr 28, 2023].
- [3] M. Cigital and W. Radosevich, "Black Box Security Testing Tools," 2015.
- [4] B. Peter. B. Peter. BPMN: A meta model for the happy path. BPMN: A meta model for the happy path. 2010 Available: <http://econpapers.repec.org/paper/dgrumamet/2010003.htm>
- [5] R. B. Kilgore, "Testing Concurrent Software Systems," 2006.
- [6] L. Lazic and N. Mastorakis, "The use of Modeling & Simulation-based Analysis & Optimization of Software Testing," *WSEAS Transactions on Information Science and Applications*, vol. 2, (11), pp. 1918-1933, 2005. Available: <https://search.proquest.com/docview/28776796>.
- [7] N. Al Moubayed and A. Windisch, "Temporal white-box testing using evolutionary algorithms," in *Apr 2009*, Available: <https://ieeexplore.ieee.org/document/4976380>. DOI: 10.1109/ICSTW.2009.17.
- [8] G. Fraser and N. Walkinshaw, "Behaviourally Adequate Software Testing," 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation, 2012. . DOI: 10.1109/icst.2012.110.
- [9] D. Bernstein, "Writing Testable Code - DZone," 2019. Available: <https://tobeagile.com/2019/05/29/write-testable-code>.

- [10] Z. Zhang, "Automated Unit Testing of Agent Systems," 2011.
- [11] M. Tanler, "Unit Testing for Ontologies," 2007.

FINAL GRADE

90/100

GENERAL COMMENTS

Instructor

Eval of Design (20): 18

User Stories (20):17

CRC (20):17

SW Testing (20):18

Testing (20): 20

Feedback:
=====

The design was well presented and discussed with good evaluation.

Detailed discussion of testing, would have liked a little more evaluation of white and black box benefits/drawbacks.

Good examples of SW testing applied to the game application.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

