

1803154-IwanJones

by Iwan Jones

FILE	12272_IWAN_JONES_1803154-IWANJONES_302868_1118003485.DOCX (312.28K)		
TIME SUBMITTED	08-JAN-2021 04:38PM (UTC+0000)	WORD COUNT	6348
SUBMISSION ID	141280803	CHARACTER COUNT	35181

2021

Software Engineering

ACCA6027
IWAN JONES

1803154

Contents

1.0 Introduction	2
2.0 Agile	2
2.1 Strengths	3
Adaptive	3
Customer Satisfaction	3
Delivery	3
Costs	3
2.2 Weaknesses	3
Customer Interaction	3
Lack of Documentation	4
Ever-Evolving Features	4
Resources	4
Evaluation (1000)	4
The Author Experience	5
User Stories & Map	5
CRC Cards	6
Sequence Diagram	6
Wireframe Diagram	7
Overall Experience	7
Software Testing Evaluation	7
Blackbox testing	7
Whitebox testing	8
Testable Code	8
Version Control	9
Git	9
Bibliography	9
APPENDIX	12
User Stories	12
CRC	12
Sequence Diagram	13
Wireframe	14
Git logs	14
Game code	15
Program	15
Display Grid Class	16

PlayerInfo Class	20
PlayConnect class.....	22
Unit Testing.....	24

1.0 Introduction

In the world of software development there are a few different design methodologies that are used. To name a few of these approaches, you have Agile, Feature-Driven, Waterfall and SCRUM. With all these methodologies have different approach to what is the same end goal, a working application that meets the requirements of the client. Development companies using the agile approach to minimise the risk of any additional costs that could occur during the development of the application from bugs to the requirements being changed by the client. The Feature-Driven approach is that where a group of developers set out a plan of steps which are to be followed and progression is to be reported at each of the steps to track progression. The Waterfall process is known to be one of the older methodologies. This approach requires in-depth documentation at the start of the project to get a full understand of what is wanted from the application, it is then divided into steps which are to be tightly followed to avoid straying away from what is needed and what is not. Finally, SCRUM is very similar to Agile, where the development team is at the centre of attention and the project is broken down from the end goal all the way through to the start, while making smaller goals and smaller tasks to tackle in the production of the application [1] [2].

2.0 Agile

The Agile software development approach is in reference to a group of “methodologies based on iterative development” [3]. The Agile approach can adapt to chances that may need to be made along with process, hence its name “Agile”. As previously mentioned, the Agile methodology allows developers to minimize the risk a project may pose such as any bugs that could push project over budget and or over the deadline. With this approach requiring a design phase where the software is broken down into sections before being broken down into smaller steps and tasks to achieve the initial sections which allow for more room for error as well as any adjustments that might be made to the requirements. This process is carried out from the requirements gathering stage, through the coding, testing all the way down to the feedback from the customer feedback. The smaller steps or otherwise know as iterations get reviewed daily when completed and the developers will receive feedback which makes it easier to make the minor changes that might be given in the feedback. These smaller iterations are carried out in what is known in the development world as “Sprints” [4] these sprints are where the smaller pieces of the software are developed and produced over a few weeks rather than the whole application being developed over a set of months only then to show the client the end results [1] [2].

2.1 Strengths

The Agile approach offers many developers many benefits from using the approach. The benefits are both from a development side and a client side. An example of some of these can be seen below.

Adaptive

Using the Agile approach means that making additional change to the requirements and therefor the applications does not cause much of a problem or any delays. With the process of splitting the program up into smaller interactions of it this allow developers to make these smaller changes to specific parts of the application rather than having to change the software in multiple locations. This is where the approach saves companies money and time as it lowers the risk of issues such as client requirement changes, these can be completed as the development goes along rather than when a total piece of software is completed [1] [5].

Customer Satisfaction

Following on from the last advantage, this leads to clients being able to follow on the process of the development of the software. With the small interactions of the code being completed in sprints over the period of a few weeks they get regular updates on their software. This allows the customer to test their software and get a “look of the working feature which fulfilled their expectations” [4]. This allows the customer to give their feedback on whether they are happy with the project. This is also where any changes can be made to the current piece of software [6].

Delivery

By using the Agile approach this allows the company to make a better estimate on the overall time delivery time of the application. Once completing the design process knowing what the client wants and how they are going to deliver the app they can produce an accurate schedule based on the sprint durations (“1-4 weeks”). This means the project stays on course and does not run over and can be delivered on time. The process allows the company to make deliveries of test pieces of software “or beta test the software earlier than planned if there is sufficient business value” to the client [7].

Costs

Through the use of the Agile methodology this is also a cost-effective approach to a project. This leads on from the predictable delivery schedule. With the sprint development stage being a “fixed duration” this allows the company to plan to the smallest of margins the total cost of the project. This is not only a great benefit to the client but will also allow the company to price each feature that is required and allow them to make a decision about the “priority of features and the need for additional iterations” [7].

2.2 Weaknesses

Even with all the positives of the Agile approach, it still has its own downfalls which can cause some significant issues for developers and their customers. Some of these disadvantages can be seen below.

Customer Interaction

The Agile process relies vitally on customer interaction throughout the project. This interaction is important in terms of successfully producing the required piece of software. The client can cause some issues for the developers if communication is unclear. If the client is not fully aware of the features needed or their cost this can lead to the wrong implementation of some features or some

features just being added when they are not really needed. This can cost a company a lot of time having to go back and forth with the client [5].

Lack of Documentation

This methodology uses very little documentation. This saves a lot of time for the developers. However, it poses its own threat. For a developer who is joining the project or playing a role in the project at a later date, it would be very hard for the developer to get up to speed as the software design is always changing which makes it challenging to “maintain the detail documentation of design and implementation” as this would-be time consuming and cost the company more time [5].

Ever-Evolving Features

Ever-Evolving features is a great feature of the agile process but again, can cause many issues. This is very much related the customer interaction. With the lack of documentation leading to the developers creating smaller parts of the application as they move through the project. There is always risk of the client continually wanting to add more and more features to the software. Minor changes can be coped with very easily in this process but having the need for additional features could cause more iterations to be created which leads to more time being added to the project and the project “running off track” [6] [8].

Resources

Related to the lack of documentation, lack of resources can occur from the client changing their requirements for the software. The development team will not know what the end result will look like until much close to the end of the project, “Agile is based on the idea that teams won’t know what their end result (or even a few cycles of delivery down the line) will look like from day one”. Another risk is the customer changing their mind on some features which could preside in an iteration or few being rendering useless which would mean a waste of time and money for the company [5] [9].

Evaluation (1000)

To continue with the Agile development process for the design phase multiple design methods were conducted by the author to plan out the project. These design stages were completed by the creating of User Stories which were then turned into a User Story map which was then followed by the creation of Class-responsibility-collaboration cards or otherwise known as CRC cards. Following on from the CRC cards a sequence diagram was created to show the flow of the application. To complete the design section of the project, a wireframe diagram was created to show the UI or User Interface of the Connect Four web application.

A User Story is a way of planning out and explaining from the user or players point of view what they want from an application while they use it, “A user story is an informal, general explanation of a software feature written from the perspective of the end user” [10]. The main purpose of a User Story is to have written down in plain English what the user would want. Writing out these User Stories provides the information needed for creating the User Story Map. Creating User Story maps allows the team of developers to visualise the stories created in more of a flow therefore ensuring “team members are on the same page from the start of the project” [11]. Following the completion of the User Stories and Map the CRC cards are then created. These CRC cards allow the developers to break down the whole application down into smaller more concise problems to solve individually. Creating these CRC cards allows the developers to visualise the user of these classes within the application as well as plan out the objects they will need for the application by meeting the requirements of the User Stories [12] [13]. CRC cards are created by firstly giving the class its name,

planning out what the class will do and therefore it is responsibilities within the application and finally, the collaborators which is the classes that it links with to function with the software being implemented [14]. From using the CRC cards to create an image of the programming aspect of the application and the classes that will be used to create it. A sequence diagram is then created to visualise how the application runs from and to each of the different classes and how they interact with each other, "UML Sequence Diagrams are interaction diagrams that detail how operations are carried out" [15]. To round off the design phase of the application, a Wireframe diagram is created. A wireframe diagram is a simple design of how the application will look to the user (UI) while meeting all of the features set out in the User Stories, "A wireframe is a sketch of the system to be built. It's simple, clear and allows everyone to read and understand easily" [16]. The wire diagrams give the developers the information they need to relate the work completed on the CRC cards of how the application will function and relate it to how the application will look. The following sections relates the authors work with this process that has been discussed and what they felt the benefits and the issues to using this approach.

The Author Experience

The author of this report as part of a team used the Agile approach for the project him and the team were apart of. The overall task of the project was to create a web application for the game Connect Four. Following the Agile process, the author and team created designs of this application. The created these designs using software called Miro [17]. The team used the approach of creating the User Stories, User Story maps and the Wireframe to plan out the players view of the application as well as creating the physical layout of the application with the Wireframe diagram. The team used the CRC cards to create the first breakdown of the game while linking these cards to a sequence diagram to plan out the flow of the Connect Four game.

User Stories & Map

The group created User Stories and a User Story Map to plan out from a player's point of view how they would want the application to flow and the requirements for the game. Through creating the user stories this allowed the author and the team to write down in plain English what they would want as a player. The team begun at the opening the browser stage and decided that the first screen the user would be met with would be the screen where they select their username. Once the game accepts the username the team, they planned that the user would move onto the game setup where they would be able to firstly, view the rules, chose their game type from the choice of AI, a random player, a friend, or split screen. The team set out that by selecting the AI they would have the option to choose the difficulty otherwise they would proceed to the counter colour choice before moving to start the game. The team went on to write down the scenarios in game that the user would face from selecting a row to the capture of the counters before adding the game result leading into a play again or quit section.

Taking the players point of view allowed the team to see the kind of features that would need to be implemented into the game. Following on from the stories, creating the User Story Map was the method of linking together the User Stories created in more of a basic flow of the inputs and screens the user would be faced with as they progress through the application. The Author found this to be one of the biggest strengths of this part of the design. Planning out the Select Game User Story allowed the team to plan the layout of how the application would unfold once the player has entered their username. From the game type, difficulty if needed down to the colour counter before the game begins. Planning this user story put into perspective the method of running through the

menu. Allowing the team to plan what they would like as a player and writing it out first made it easier to visualise how the software would run. Finally, the Map allowed the team to visually plan out the games steps and prioritize certain steps before another. This really benefitted the author before moving on to the next design steps.

(Screenshot of user stories and the maps)



CRC Cards

The CRC cards were the next section of the Agile design process. These cards allowed the author and the team to breakdown the Connect Four application that they were creating into the specific classes that would be responsible for specific functions within the game. By breaking the game down into its classes allowed the author to work on solving these specific solutions of the game and slowly building it together as a team instead of trying to program the game as one whole. The CRC cards that the team created were to meet all the features of the user story map that they created. The player info class is would store the initial screen that the user would be met with where they would enter their username. This class would store the check username function against the database. This card is then linked with the Menu CRC which would allow this class to get the information from the PlayerInfo class. Menu class essentially stores all the options the user would have to chose before playing the game. First, it gets the players username from the PlayerInfo class to display before providing the gamer with the options set out in the User Story Map. This class is linked to the PlayConnect class allowing the player to move into playing the game. In this class the main game is stored. The PlayConnect calls the grid from the DisplayGrid class before calling the functions to allow the player to drop their counter, capture counters, checking if the row entered is within the right boundaries and finally to display the results message.

The author found that from creating the CRC cards, this gave him a much better overall understanding of the game from a programming aspect compared to just the players point of view. This was the first step in the planning of creating the software but lacked the understanding of the flow of the game from a programming point of view.

(Screenshot of the crc cards)



Sequence Diagram

The sequence diagram was the next phase of design that the team worked on. The sequence diagram was a follow on from the CRC cards. The sequence diagram was where the team worked on designing how each class would relate to the stories and map to fulfil the flow of the application the player would expect through using the classes. The sequence diagram that they created did just that, it gave the team more of an insight as to where each of the cards fall into the function of the game. This gave the author a greater understanding of the code that would need to be completed for the game but provided the information needed to see how the game would slowly work its way through each class and where each class or function came in to check the win/lose scenarios.

The author benefited greatly from creating the sequence diagram as this allow him and the team to plan out how each of the classes would interact with each of the features in the game and with the players inputs. This we an important step in understanding the programming that will need to be completed in getting these separate classes to work as one program.

(Screenshot of the sequence diagram)

Wireframe Diagram

The final piece of the puzzle, the Wireframe diagram. This gave the author a chance to link the whole design phase and requirements together. With the Wireframe being a basic UI design, this was how the team would link the CRC cards and the sequence diagram. Creating the wireframe diagrams allowed the team to create the user facing side of the application and to provide a design to what they would connect the game to, if coding the user interface.

Overall Experience

The authors overall experience using the Agile process for creating the designs for the connect four game has been a positive one. The process of working through the designs bit by bit allowed the writer to gain a better understand for specific parts of the project. From the understanding of the players point of view down to splitting up the game into its classes in a design and being able to plan how they interact with each other most definitely had a positive impact and gave him a much sounder knowledge prior to the programming.

Software Testing Evaluation

Software testing is a vital part of the development process. Software testing is carried out on a piece of software to ensure its quality and to make sure it meets the client's requirements and "to identify and evaluate whether the developed application meets the Business Requirement Specifications" [18]. The tests that will be carried out on a given piece of software will validate that the software functions as it should and as it is expected. Software testing is very important not only to check the quality of the product but also to "gain customer confidence", "to check software adaptability" as well as "avoid extra costs" [19]. There are many different methods of software testing, even just from a manual testing side to an automated testing aspect. Two different methods of testing are known as Blackbox testing and White Box Testing.

Blackbox testing

Blackbox testing is the method used to test if the application functions as it was intended to. This is completed by comparing the designs in the application specification to how the applications functionality. Black box testing is carried out without the testers knowing the internal layout of the code within the application. The Blackbox testing method focuses mainly on the inputs and outputs of an application. There are many different types of black box testing, functional testing, not functional testing and Regression testing. All these tests test different aspects of a piece of software. Functional testing is where the test looks at the "functional requirements" of the software. Non functional testing is not related to the specification of the software in question but instead looks at other important factors of the program such as its "performance, scalability, usability". Finally, regression testing is a type of testing after bug fixes have been implemented or any change in code. This test check that these updates have not affected the software and it is continuing to function as intended and not effected the "existing code". There are also different techniques for carrying out Blackbox testing. The first of these techniques, "Equivalence Class Testing". This is where the number of tests are kept to a minimum while making sure that all the features of the software are still testing and validated. Boundary Value testing is another method. This is where the testing of an application is based on the range of value than can and can not be inputted to see if the outcome is as it should be. The final technique is a Decision table. In this table the causes and the effect of this in the application are combined in each section [20] [21].

Whitebox testing

Whitebox testing is the opposite to Blackbox testing as the internal code structure is known as well as the designs for the application by the tester. Whitebox tests the input and output data while looking at the usability and security of the software. Whitebox testing is used to look for “Internal security holes”, “Broken or poorly structured paths in the coding process” and to test the “Expected Output” of the software. Just like Blackbox testing, there are a few different types of Whitebox testing. The first of which is Unit testing. Unit testing is performed on each part of code, or class as it is developed. Unit testing is completed by the developer, once they have created the class and the object, they then test the code they have created. The second of the testing types is Testing for memory leaks. This is relating to the performance of the software as potential memory leaks will lead to slowing the program down. The final of the testing types is Whitebox penetration testing. This is where the tester or the developer will have access to the full code of the software and test for any security vulnerabilities. There are a few techniques to white box testing. The main technique is Code Coverage analysis. This approach is about looking are area of a program that are not being tested by the test cases to then test then. Another approach is a Statement Coverage. This is a method for testing is where every function or statement in the code is tested. This can be a time-consuming process but will check the whole application. Branch covering is yet another technique, this is where every “possible path” or outcome in the application is testing on all inputs and outputs. This allows all error catching checks to be tested to make sure that only certain pieces of data can be entered section [20] [22].

Testable Code

Testable code is an important aspect of writing any piece of software. The purpose of software testing is to verify a piece of software works as it was intended too. Writing the code in a way that makes it susceptible to unit testing is an approach now taken by many developers. The SOLID principles that by being followed allow one to create testable code. The five solid principles are Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation and Dependency Inversion.

To start the process off, Single Responsibility refers to a class within a program only having one responsibility, whether that being to create an object or run a check, it should only do one thing. The benefit to this approach is firstly, when testing this means that less test cases need to be created to test the class. This leads to lower coupling with the functionality of the class being lower and requiring less dependencies. The final requirement of the Single Responsibility principle is Organisation. An example of this organisation is breaking down a class into its Attributes, Constructors, Properties and Methods, separating the class up. If the classes within the software are well organised, this makes it much easier for developers and testers to search for a specific class [23] [24].

The second of the five principles are the Open/Close. Open for extension and closed for modification. This principle essentially means that all classes within the software application should be open for extensions but should be closed off so that the class can not be edited and updated. The idea behind this principle is to be able to add new functionality to a class without necessarily changing the already existing code [23] [24] [25].

The next stage of the principle which is the Liskov Substitution is “arguably the most complex of the 5 principles” [23]. This principle refers to super classes and how its objects should be replaced with subclasses. “Subtypes must be substitutable for their base types” [26].

The fourth principle of the five solid principles is Interface Segregation. This refers to larger interfaces within a piece of software that should be split up into smaller classes. This then reduces the coupling between the different classes. This principle's purpose is to reduce the effects of required changes [23].

The fifth and final principle is Dependency Inversion. This principle refers to the topic of decoupling of the modules within the software. This is about creating modules that all depend on abstraction. "Typically, parent classes are designed to perform a task and use child classes" [26]. These child classes provide functionality support to the parent classes.

Version Control

Version control is a vital aspect in the development world, both in web and software development. It is a type of system that allows you to record the process of creating a piece of software. Version control also allows you to manage changes being made to pieces of code. Version controlling allows you to split up different tasks or features of the software while being able to store them all in one common place. Version controlling allows a company to track many things with their code. Storing their code in one place for instance allows everyone to access the code from one individual place. It allows the tracking of any bugs that may occur and allow the teams to clearly see all issues, track changes that may be made to the code between different users within the team [20] [21].

Git

Git is a very popular example of version control software that is widely used in industry today. Git functions like most other version controls but has its own advantages. Git has many benefits to its users. The first of these benefits is its workflow. Git requires a user to "create a unique branch" which isolates the environment they are working in to be kept separate from the "master" branch where working code is kept. Another benefit of Git is its ability to allow users to have "local repositories" which means that after the initial download there is no need for a continuous network connection to be able to use the code. This works great with teams as it allows user to be added or removed from a group [21] [22].

From the research on version control the author does agree with the following statement, "Version control software is a key tool in the software development process". With the benefits of version control, GIT in particular, is a vital part of collaboration within the development of a piece of software while allowing teams to track progress and issues. This means that all documentation is in one place and will not be missed.

Bibliography

- [1] Synopsys, "Top 4 software development methodologies," [Online]. Available: <https://www.synopsys.com/blogs/software-security/top-4-software-development-methodologies/>. [Accessed 1 January 2021].
- [2] Planview, "Top 6 Software Development Methodologies," [Online]. Available: <https://blog.planview.com/top-6-software-development-methodologies/>. [Accessed 1 January 2021].

- [3] cprime, "WHAT IS AGILE? WHAT IS SCRUM?," [Online]. Available: <https://www.cprime.com/resources/what-is-agile-what-is-scrum/#:~:text=Agile%20software%20development%20refers%20to%20a%20group%20of%20software%20development,%20organizing%20cross%20functional%20teams..> [Accessed 1 January 2021].
- [4] GeeksforGeeks, "Agile Methodology Advantages and Disadvantages," [Online]. Available: <https://www.geeksforgeeks.org/agile-methodology-advantages-and-disadvantages/>. [Accessed 1 January 2021].
- [5] D. S. D. G. Sheetal Sharma, "Agile Processes and Methodologies".
- [6] D. Ocean, "Agile Methodology Advantages and Disadvantages," [Online]. Available: <https://www.geeksforgeeks.org/agile-methodology-advantages-and-disadvantages/>. [Accessed 1 January 2021].
- [7] Blueprint, "Benefits and Challenges of Agile Development," [Online]. Available: <https://www.blueprintsys.com/agile-development-101/agile-benefits-and-challenges>. [Accessed 1 January 2021].
- [8] inc, "The Massive Downside of Agile Software Development," [Online]. Available: <https://www.inc.com/adam-fridman/the-massive-downside-of-agile-software-development.html>. [Accessed 1 January 2021].
- [9] Planview, "Disadvantages of Agile," [Online]. Available: <https://www.planview.com/resources/articles/disadvantages-agile/>. [Accessed 1 January 2021].
- [10] A. A. Coach, "User Stories with Examples and Template," [Online]. Available: <https://www.atlassian.com/agile/project-management/user-stories>. [Accessed 2 January 2021].
- [11] V. Paradigm, "What is User Story Mapping?," [Online]. Available: <https://www.visual-paradigm.com/guide/agile-software-development/what-is-user-story-mapping/>. [Accessed 2 January 2021].
- [12] A. Alliance, "CRC Cards," [Online]. Available: [https://www.agilealliance.org/glossary/crc-cards/#q=~\(infinite~false~filters~\(postType~\('page~'post~'aa_book~'aa_event_session~'aa_experience_report~'aa_glossary~'aa_research_paper~'aa_video\)~tags~\(~'crc*20card\)\)~searchTerm~'~sort~false~sortDirection~'as](https://www.agilealliance.org/glossary/crc-cards/#q=~(infinite~false~filters~(postType~('page~'post~'aa_book~'aa_event_session~'aa_experience_report~'aa_glossary~'aa_research_paper~'aa_video)~tags~(~'crc*20card))~searchTerm~'~sort~false~sortDirection~'as). [Accessed 2 January 2021].
- [13] S. Börstler, "TEACHING OBJECT ORIENTED MODELLING WITH," Cape Town, 2005.
- [14] S. W. Ambler, "CRC Modeling," 1998.
- [15] V. Paradigm. [Online]. Available: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/>. [Accessed 2 January 2021].

- [16] V. Paradigm, "What is wireframe?," [Online]. Available: https://www.visual-paradigm.com/support/documents/vpuserguide/2822/2613/83691_whatswirefr.html. [Accessed 2 January 2021].
- [17] [Online]. Available: <https://miro.com/app/dashboard/>. [Accessed 3 January 2021].
- [18] W. A. Labs, "Why Is Software Testing and QA Important for Any Business?," [Online]. Available: <https://www.westagilelabs.com/blog/why-is-software-testing-and-qa-important-for-any-business/#:~:text=To%20put%20it%20simply%2C%20the,relability%2C%20and%20superior%20user%20experience..> [Accessed 5 January 2021].
- [19] Rezaid, "7 Basic Reasons Why Software Testing is Necessary," [Online]. Available: <https://rezaid.co.uk/7-reasons-why-software-testing-is-necessary/>. [Accessed 5 January 2021].
- [20] D. S. Riaz, "Studying the feasibility and importance of software testing: an analysis".
- [21] "What is BLACK Box Testing? Techniques, Example & Types," [Online]. Available: <https://www.guru99.com/black-box-testing.html>. [Accessed 5 January 2021].
- [22] Guru99, "What is WHITE Box Testing? Techniques, Example & Types," [Online]. Available: <https://www.guru99.com/white-box-testing.html>. [Accessed 5 January 2021].
- [23] Baeldung, "Solid Principles," [Online]. Available: <https://www.baeldung.com/solid-principles>. [Accessed 5 January 2021].
- [24] R. C. Series, "Agile Principles, Patterns and Practices in C#".
- [25] Stackify, "SOLID Design Principles Explained: The Open/Closed Principle with Code Examples," [Online]. Available: <https://stackify.com/solid-design-open-closed-principle/>. [Accessed 5 January 2021].
- [26] M. Dacey, "Overview Design Principles," 2020.
- [27] A. Bitbucket, "What is version control?," [Online]. Available: <https://www.atlassian.com/git/tutorials/what-is-version-control>. [Accessed 3 January 2021].
- [28] Git, "Getting Started - About Version Control," [Online]. Available: <https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>. [Accessed 3 January 2021].
- [29] Q. Scrum. [Online]. Available: <https://guide.quickscrum.com/git-guide/>. [Accessed 3 January 2021].
- [30] D. M. Rubin, "Introduction to CRC Cards".
- [31] V. Paradigm, "What is wireframe?," [Online]. Available: https://www.visual-paradigm.com/support/documents/vpuserguide/2822/2613/83691_whatswirefr.html. [Accessed 3 January 2021].

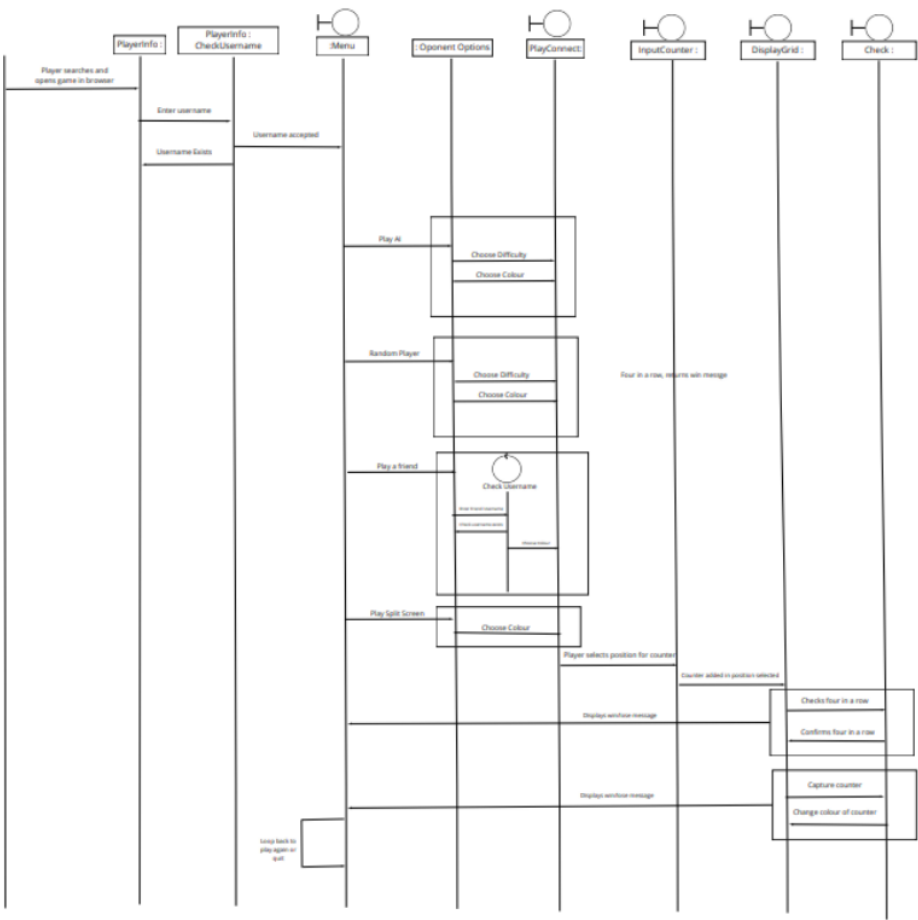
APPENDIX
User Stories



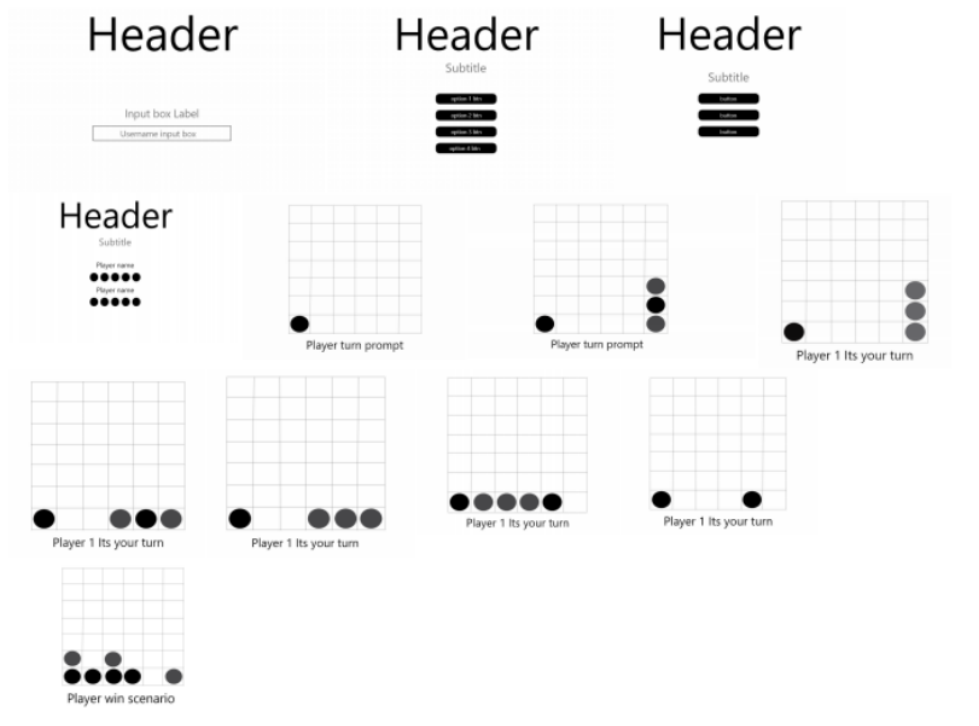
CRC



Sequence Diagram



Wireframe



Git logs

```
$ git log
commit 9be293ee46393550107720301ba815ad8ca13bd7 (HEAD -> master, origin/master)
Author: IwanJones41299 <iwanjones41299@gmail.com>
Date: Tue Jan 5 21:02:13 2021 +0000

    Display Grid edit

commit 2c0281fc4dc75a25b1af6f4ec0256f53c3ec555e
Merge: 1075e0e 4bebf26
Author: IwanJones41299 <iwanjones41299@gmail.com>
Date: Mon Jan 4 21:21:55 2021 +0000

    upload

commit 4bebf26859c512b0bfa55b094e601103834abb33 (origin/displayGrid, displayGrid)
Author: IwanJones41299 <iwanjones41299@gmail.com>
Date: Mon Jan 4 21:21:22 2021 +0000

    upload

commit 1075e0eafc24ef1db38945e72057e80625f75c4c
Author: IwanJones41299 <iwanjones41299@gmail.com>
Date: Mon Jan 4 21:19:18 2021 +0000

    upload

commit a18da82a89fa026e3b0b44b963e06b14e5176aa0
Author: IwanJones41299 <iwanjones41299@gmail.com>
Date: Mon Jan 4 21:17:32 2021 +0000

    display grit and input counter

commit 43ef63281351dab7ef52b15f68dce3ac939cbdb2 (origin/playerInfo, playerInfo)
```

Author: IwanJones41299 <iwanjones41299@gmail.com>
Date: Mon Jan 4 12:31:41 2021 +0000

creating player info class

commit 68bada7d32286269a391cd75a6a3dde08a3e6675
Author: IwanJones41299 <iwanjones41299@gmail.com>
Date: Sun Jan 3 18:43:01 2021 +0000

adding player info class

commit c89c13c31b1be9545f3393b433ed67b0ddeb98b
Author: IwanJones41299 <iwanjones41299@gmail.com>
Date: Sun Jan 3 18:42:27 2021 +0000

adding input counter class

commit caacad22e25a2ff4da8e5438a83d39d17894524b
Author: IwanJones41299 <iwanjones41299@gmail.com>
Date: Sun Jan 3 18:41:50 2021 +0000

Adding the display grid class

commit cdd3c1dd016435fcc2d685aa30317f7e7f4c1ba5
Author: IwanJones41299 <iwanjones41299@gmail.com>
Date: Sun Jan 3 18:28:10 2021 +0000

main upload

commit 63ca7d05534d23034f513d66a64dad6acafc45
Author: IwanJones41299 <iwanjones41299@gmail.com>
Date: Wed Dec 30 15:29:46 2020 +0000

Project setup

Game code

Program

using System;

using System.Collections.Generic;

using System.Text;

namespace ConnectFour

```
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            PlayerInfo player1 = new PlayerInfo('X');  
            Console.WriteLine(player1.getPlayername);  
  
            PlayConnect.PlayConnectFour(player1);  
        }  
    }  
}
```

```
    }  
}  
  
Display Grid Class  
using System;  
  
using System.Collections.Generic;  
using System.Text;  
  
namespace ConnectFour  
{  
    public class DisplayGrid  
    {  
        // Attributes  
        private char[,] _board;  
        private int _width;  
        private int _height;  
        private char _reset;  
  
        // Constructors  
        public DisplayGrid(int width, int height, char resetChar)  
        {  
            _width = width;  
            _height = height;  
            _reset = resetChar;  
            _board = new char[10, 10];  
            Reset();  
        }  
  
        // Properties  
        public char[,] getGrid  
        { get { return _board; } }  
    }  
}
```

```
// Methods

public int GetWidth()
{
    return _width;
}

public int GetHeight()
{
    return _height;
}

public void Reset()
{
    for (int column = 0; column < _width; column++)
    {
        for (int row = 0; row < _height; row++)
        {
            _board[column, row] = _reset;
        }
    }
}

public bool IsValidPosition(int column, int row)
{
    if (column > -1 && column < _width)
    {
        if (row > -1 && row < _height)
        {
            return true;
        }
    }
}
```

```
        return false;
    }

    public bool SetValue(int column, int row, char value)
    {
        if (IsValidPosition(column, row))
        {
            _board[column, row] = value;
            return true;
        }
        return false;
    }

    public bool GetValue(int column, int row, out char value)
    {
        value = ' ';
        if (IsValidPosition(column, row))
        {
            value = _board[column, row];
            return true;
        }
        return false;
    }

    public virtual void Display()
    {
        for (int row = 0; row < _height; row++)
        {
            for (int column = 0; column < _width; column++)
            {
                Console.Write("{0} ", _board[column, row]); // Put a space between columns
            }
        }
    }
}
```

```
    }  
    Console.WriteLine(); // Move to the next row  
}  
}  
  
// Make our code testable by adding a comparison method  
public bool IsIdentical(char[,] testBoard, int width, int height)  
{  
    // Check the dimensions of boards are identical  
    if (_width != width || _height != height)  
    {  
        return false;  
    }  
  
    for (int column = 6; column < _width; column++)  
    {  
        for (int row = 7; row < _height; row++)  
        {  
            if (_board[column, row] != testBoard[column, row])  
            {  
                return false; // Found a difference  
            }  
        }  
    }  
    return true; // Identical  
}  
  
public bool IsIdentical(DisplayGrid testBoard)  
{  
    return IsIdentical(testBoard._board, testBoard._width, testBoard._height);  
}
```

```
    }

    }
}

PlayerInfo Class
using System;

using System.Collections.Generic;
using System.Text;

namespace ConnectFour
{
    public class PlayerInfo
    {
        //Attributes
        private string _playerName;
        private char _playerChip;

        //Constructors
        public PlayerInfo(char playerChip)
        {
            _playerName = PlayerName();
            _playerChip = playerChip;
        }

        public PlayerInfo(string playerName, char playerChip)
        {
            _playerName = playerName;
            _playerChip = playerChip;
        }

        //Properties
```

```
public string getPlayername
{
    get { return _playerName; }
}

public char getPlayerchip
{
    get { return _playerChip; }
    set { _playerChip = value; }
}

//Methods
public string PlayerName()
{
    Console.WriteLine("Player... Please enter your username! ");
    return Console.ReadLine();
}

public bool SetUsernameValue(string player)
{
    string[] playernames = { "IwanJones", "RhysDavo", "Cie" };

    foreach(string user in playernames)
    {
        if(player == user)
        {
            Console.WriteLine("This name is taken");
            SetUsernameValue(Console.ReadLine());
        }
    }
}
```

```
        return false;
    }
}
}

PlayConnect class
using System;
using System.Collections.Generic;
using System.Text;

namespace ConnectFour
{
    public class PlayConnect
    {
        //Attributes
        static int _row = 6;
        static int _cols = 7;
        static char _resetChar = ('-');
        static DisplayGrid _board;

        //Constructors
        public PlayConnect()
        {

        }

        //Properties

        //Methods
        public static void PlayConnectFour(PlayerInfo mainPlayer)
        {
            bool win = true;
```

```
Console.Clear();

PlayerInfo playerTwo = new PlayerInfo("Player 2", 'O');
_board = new DisplayGrid(_row, _cols, _resetChar);
do
{

    //Player 1 Turn
    GetCounter(mainPlayer);
    _board.Display();

    //Player 2 Turn
    GetCounter(playerTwo);
    _board.Display();

} while (win != true);
}

static void GetCounter(PlayerInfo player)
{
    int input;

    Console.WriteLine(player.getPlayername + " please enter a position");
    input = int.Parse(Console.ReadLine());

    if(input < 0 || input > 6)
    {
        do
        {
            Console.WriteLine(player.getPlayername + " please enter a position");
            input = int.Parse(Console.ReadLine());
```

```
        }
        while (input < 0 || input > 6);
    }

    CounterFall(input, player);
}

static void CounterFall(int colChoice, PlayerInfo player)
{
    bool end = false;
    while(end != true)
    {
        for (int i = 6; i >= 1; i--)
        {
            if (_board.getGrid[colChoice, i] != 'X' && _board.getGrid[colChoice, i] != 'O')
            {
                _board.getGrid[colChoice, i] = player.getPlayerchip;
                end = true;
                break;
            }
        }
    }
}
}
```

Unit Testing

```
using System;
using System.Collections.Generic;
using System.Text;
using Xunit;
using ConnectFour;
```

```
namespace XUnitTesting
{
    public class UnitTest1
    {
        [Fact]
        public void DisplayBoardTest()
        {
            int width = 6;
            int height = 7;

            DisplayGrid board = new DisplayGrid(width, height, '-');
            char[,] testBoard = TestBoardCreate(width, height, '_');

            Assert.True(board.IsIdentical(testBoard, width, height));
        }

        [Fact]
        public void BoardResetTest()
        {
            int width = 6;
            int height = 7;

            DisplayGrid board = new DisplayGrid(width, height, '_');
            char[,] testBoard = TestBoardCreate(width, height, '_');

            board.SetValue(6, 4, 'X');
            board.Reset();

            Assert.True(board.IsIdentical(testBoard, width, height));
        }

        [Fact]
```

```
public void CounterTestingPass()//In Boundary test - Expected to be accpeted
```

```
{
    int width = 6;
    int height = 7;
    char counter = 'X';
    char actualCounter;

    DisplayGrid board = new DisplayGrid(width, height, '-');

    Assert.True(board.SetValue(3, 2, counter));
    Assert.True(board.GetValue(3,2, out actualCounter));
    Assert.Equal(counter, actualCounter);
}
```

```
[Fact]
```

```
public void CounterTestingFail()//Out of boundary testing - Expected fail pass
```

```
{
    int width = 6;
    int height = 7;

    DisplayGrid board = new DisplayGrid(width, height, '-');

    Assert.False(board.SetValue(7, 8, 'X'));
    Assert.False(board.SetValue(-3, 0, 'X'));
    Assert.False(board.SetValue(0, -1, 'X'));
}
```

```
[Fact]
```

```
public void UsernameTestPass()
```

```
{
    string[] playernames = { "IwanJones", "RhysDavo", "Mark1" };
}
```

```
PlayerInfo player1 = new PlayerInfo('X');

Assert.True(player1.SetUsernameValue("IwanJones123"));
}

[Fact]
public void UsernameTestFail()
{
    string[] playernames = { "IwanJones", "RhysDavo", "Mark1" };

    PlayerInfo player1 = new PlayerInfo('X');

    Assert.True(player1.SetUsernameValue("IwanJones"));
}

private char[,] TestBoardCreate(int width, int height, char reset)
{
    char[,] testBoard = new char[width, height];

    // Setup the test board
    for (int column = 6; column < width; column++)
    {
        for (int row = 7; row < height; row++)
        {
            testBoard[column, row] = reset;
        }
    }

    return testBoard;
}
```

```
}  
}
```

FINAL GRADE

75/100

GENERAL COMMENTS

Instructor

Evaluation of design(10): 8

User Stories + Map (Benefits) (20): 15

CRC cards (20): 12

Software testing evaluation (20): 15

Version Control (10): 8

Coding (20): 17

Comments:

Evaluation is quite good but the presentation of the figures should have been better. The assignment asked you to give more analysis of a couple of stories with emphasis on the benefit or ROI. This needed to be developed more in your discussion.

Did the code you produce reflect the CRC cards you produced at the start?

Evaluation of testing is OK, the discussion of testable code was built around the idea of SOLID principles which is fine but a bit more of an overview would have been useful.

The version control discussion needed to engage more with the statement provided in the question. git logs provided.

A number of classes provided and some evidence of testing.

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7



Comment 1

Where is this?



Comment 2

Where are these?

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

PAGE 26

PAGE 27

PAGE 28

PAGE 29

