System Verification and Validation Plan for SFWRENG 4G06

Team 9, dice_devs
John Popovici
Nigel Moses
Naishan Guo
Hemraj Bhatt
Isaac Giles

November 4, 2024

Revision History

Table 1: Revision History

Date	Developer(s)	Change
2024-10-27	John Popovici	Preparing file and added 1.1, 1.2, 1.3
2024-10-28	John Popovici	Removed table of symbols; Added 1.4
2024-10-30	Hemraj Bhatt	Added content to section 2
2024-10-31	John Popovici	Finalized section 1 formatting and content
2024-11-01	Isaac Giles	Filled in content for section 3
2024-11-02	Naishan Guo	Added new content for section 3
2024-11-04	Hemraj Bhatt	Added content to reflection
...	...	...

[The intention of the VnV plan is to increase confidence in the software. However, this does not mean listing every verification and validation technique that has ever been devised. The VnV plan should also be a **feasible** plan. Execution of the plan should be possible with the time and team available. If the full plan cannot be completed during the time available, it can either be modified to "fake it", or a better solution is to add a section describing what work has been completed and what work is still planned for the future. —SS]

[The VnV plan is typically started after the requirements stage, but before the design stage. This means that the sections related to unit testing cannot initially be completed. The sections will be filled in after the design stage is complete. the final version of the VnV plan should have all sections filled in. —SS]

Contents

List of Tables

[Remove this section if it isn't needed —SS]

List of Figures

[Remove this section if it isn't needed —SS]

1 General Information

1.1 Summary

This project creates an online multiplayer game platform that allows for the creation of custom Yahtzee-like games and variants. This family of games will come with some presets such as classic Yahtzee, an octahedron version, and more, but will allow for the users to set their own variables such as number of dice, what kind of dice, and some elements of scoring, and then play that game. Kinds of dice would include cubes and octahedrons, among other multi-sided dice, and scoring could be calculated at the end of the game, as in classic Yahtzee, or on a per-round basis, where hands go in a head-to-head matchup.

1.2 Objectives

The main objective of this project is to ensure robust and flexible functionality that supports customizability in game creation, with confidence in their interactability. Specifically, the project will focus on:

- Demonstrating Adequate Usability: User interactions should be intuitive, enabling smooth navigation and game setup without requiring extensive technical knowledge.
- Supporting User-Driven Customizability: Emphasis will be placed on making the platform adaptable, allowing users to easily create and modify games with various dice types and scoring rules. These elements must be modular such that a custom game can make use of these components in a flexible fashion.
- Ensuring Software Correctness: Testing will confirm accurate dice roll usage, score calculations, and game outcomes to build confidence in the correctness of the core mechanics.

Out of Scope Objectives: Due to time constraints, certain areas are out of scope for this project. These include:

- Probability Testing: While a statistical analysis could be done on the randomness of dice rolls, it would be out of scope for our current plans

and we will proceed with a general assumption that functions that rely on random values are as accurately pseudo-random as needed, without set seeding.

- Performance Optimization: While performance will be monitored, extensive optimization for faulty connections or intricate rule configurations will be left out, with plans to address this in potential future releases.

This prioritization allows us to allocate resources efficiently, focusing on core functionality and foundational elements.

1.3 Challenge Level and Extras

Through the implementation of not just a single Yahtzee game variant, but by implementing a system where the user can create a custom game variant and connect to another online player to play it, we are looking to develop our project to be both a fun game and a capstone project. The challenge difficulty approved is that of general.

We will include extra elements to aid in better developing our game and allow us to design for a fun gaming experience. Following industry practices in game development, we will do the following extras:

- Focus groups, Surveys, and Interviews, focused on Usability Testing
- User documentation

1.4 Relevant Documentation

The following documents are crucial for the Verification and Validation process, ensuring the project meets its objectives and operates reliably:

- Software Requirements Specification (SRS): This document outlines the core requirements for the platform, defining features and limitations, as well as expected behaviors. The SRS serves as a primary reference to validate that the developed software meets user and system requirements.
- Hazard Analysis (HA): This document is key to identifying and managing risks associated with system failures, particularly in dice physics, scoring, and multiplayer stability.

- Module Interface Specification (MIS): The MIS details the interfaces between components, which is essential for ensuring seamless interaction across modules and verifying that dependencies are managed correctly.
- Module Guide (MG): The MG provides a structural breakdown of the system's components, allowing verification that each module aligns with the SRS specifications and functions as intended.

2 Plan

This section covers the project's verification and validation (V&V) strategy, including the methodologies and strategies used to ensure software dependability, correctness, and usability. Each component of the plan will establish verification and validation procedures appropriate for each stage of the project lifecycle, from reviewing the Software Requirements Specification (SRS) to testing

2.1 Verification and Validation Team

[Your teammates. Maybe your supervisor. You should do more than list names. You should say what each person's role is for the project's verification. A table is a good way to summarize this information. —SS]

Team Member	**Role**	**Responsibilities**
Member X	SRS Verification Lead	Oversees SRS review, organizes feedback, coordinates checklist compilation, and ensures any new or modified requirements are accurate.
Member X	Design Verification Lead	Oversees design review, develops checklists, and organizes peer review meetings regarding design. They also ensure that any design modifications that either are outcomes of meetings or decisions made within the team are properly formulated and accurate.
Member X	Implementation Tester	Performs unit tests, isolates tests for core capabilities, and ensures that individual code components are in line with the implementation plan outlined in this document. In charge of testing any new modifications and green-lighting them.
Member X	Software Validation Lead	In charge of ensuring that the software matches user requirements through using feedback provided by supervisor and peer reviews. Additionally, in charge of performing validation strategies outlined in the document.
Member X	Tool and Automation Lead	Oversees the installation and usage of automated testing tools, unit testing frameworks, and test coverage metric tracking tools. In charge of sourcing the correct tools needed for the specific system as well.
Supervisor	Project Reviewer	Provides oversight, participates in structured review sessions, and provides feedback on requirements, design, and implementation reviews. Due to the supervisor's area of expertise, feedback will focus on requirements and design as opposed to programming implementation advice.

Table 2: Verification and Validation Team Roles

2.2 SRS Verification Plan

[List any approaches you intend to use for SRS verification. This may include ad hoc feedback from reviewers, like your classmates (like your primary reviewer), or you may plan for something more rigorous/systematic. —SS]

[If you have a supervisor for the project, you shouldn't just say they will read over the SRS. You should explain your structured approach to the review. Will you have a meeting? What will you present? What questions will you ask? Will you give them instructions for a task-based inspection? Will you use your issue tracker? —SS]

[Maybe create an SRS checklist? —SS]

The SRS verification will take an organized approach, including supervisor and peer review sessions, to ensure that the stated requirements are clear, accurate, and thorough.

- **Structured Review Meeting**: Conducted with the supervisor in person to present the essential SRS components. The team will highlight the essential functional and non-functional requirements and solicit feedback on the clarity and feasibility of each requirement. The feedback will then be tracked as issues and be used to make any modifications deemed necessary.

- **Checklist Creation**: A detailed checklist will be prepared to analyze each SRS item, ensuring the requirements are accurate and feasible for execution. The checklist will be utilized at each review session. If the checklist is filled out completely, the team will know that there are no necessary changes and if it is not then necessary modifications will be made to the SRS.

- **Peer Review Sessions**: The team will have their peers such as classmates conduct task-based inspections of the SRS to identify any ambiguities or missing components. This will ensure that the SRS is complete and viable.

2.3 Design Verification Plan

[Plans for design verification —SS]

[The review will include reviews by your classmates —SS]

[Create a checklists? —SS]

To ensure that the design complies with the SRS standards, the team will conduct a structured review process including the entire team, external peers and the teams supervisor:

- **Design Checklists**: A checklist will be developed to guide reviewers through each design component and confirm whether or not they correspond to both the functional and non-functional requirements outlined in the SRS.
- **Peer Review**: Classmates and primary reviewers will use the checklist to determine if the design is consistent with the SRS. This includes the code structure, front-end design, and game mechanics.
- **Structured Review Meeting**: The team will schedule a review session with our supervisor where each design element will be presented. The goal being to receive feedback and make any necessary modifications.

2.4 Verification and Validation Plan Verification Plan

[The verification and validation plan is an artifact that should also be verified. Techniques for this include review and mutation testing. —SS]

[The review will include reviews by your classmates —SS]

[Create a checklists? —SS]

The V&V plan itself will be verified to make sure it adequately addresses every facet of the project and sufficiently plans appropriate methods of verification and validation.

- **Mutation Testing**: To guarantee robustness of our software, we'll use mutation testing to assess how well the test cases work and to make sure that the methods used for verification can accurately detect software faults.
- **Review Sessions**: To ensure that all necessary verification and validation methods are covered, the V&V plan will be reviewed by the teams peers and supervisor utilizing a checklist. The team will also independently review the methods to ensure all bases are covered.

2.5 Implementation Verification Plan

[You should at least point to the tests listed in this document and the unit testing plan. —SS]

[In this section you would also give any details of any plans for static verification of the implementation. Potential techniques include code walkthroughs, code inspection, static analyzers, etc. —SS]

[The final class presentation in CAS 741 could be used as a code walkthrough. There is also a possibility of using the final presentation (in CAS741) for a partial usability survey. —SS]

The implementation verification will primarily focus on unit testing and manual testing, emphasizing the isolation of modules and testing them with constant inputs mimicking user inputs. Given that project is a game, testing poses unique challenges and nuances that require careful consideration.

- **Unit Testing**: The team will employ isolated unit tests, setting fixed inputs to verify core functionalities like game logic, scoring systems, and user input handling. Simulations of dice rolls and the scores that go along with them are one example. Additionally, to ensure that tests only concentrate on the module's functionality and are not influenced by outside factors, mocking will be used to mimic external dependencies.

- **Manual Testing**: Manual testing will be used to evaluate the game from a player's point of view and make sure that gameplay is fluid, intuitive, and stimulating, manual testing will be used. In order to find any possible errors or discrepancies in the game mechanics, test cases will cover a range of scenarios, including edge cases and user interactions.

- **Code Walkthroughs**: In order to collect feedback on code correctness, standard compliance adherence, and alignment with design specifications outlined in the SRS, team-based code walkthroughs will be carried out. These walkthroughs will be held with our peers and external individuals with expertise in the field.

- **Static Analysis**: FxCop, a free static code analysis tool from Microsoft, will be utilized to evaluate the code. It excels at analyzing .NET managed code assemblies for adherence to Microsoft's .NET

Framework Design Guidelines. This tool will help ensure code quality and identify potential issues early in the development cycle. It is particularly well-suited for this project, as the game is being developed using C# on the .NET version of Godot.

2.6 Automated Testing and Verification Tools

[What tools are you using for automated testing. Likely a unit testing framework and maybe a profiling tool, like ValGrind. Other possible tools include a static analyzer, make, continuous integration tools, test coverage tools, etc. Explain your plans for summarizing code coverage metrics. Linters are another important class of tools. For the programming language you select, you should look at the available linters. There may also be tools that verify that coding standards have been respected, like flake9 for Python. —SS]

[If you have already done this in the development plan, you can point to that document. —SS]

[The details of this section will likely evolve as you get closer to the implementation. —SS]

To maintain code quality and reliability, teh team will implement a variety of automated testing tools. Any mentioned tools are not finalized and may be substituted based on team preferences and needs:

- **Unit Testing Framework**: The team will employ a unit testing framework, such as NUnit or xUnit, recommended by Microsoft, to automate our test cases. This will help ensure that each game feature functions consistently and accurately.
- **Test Coverage Tools**: The team will utilize Coverlet which provides a cross-platform code coverage framework for C#. As part of the .NET Foundation, Coverlet is recommended by Microsoft and will help us track coverage metrics to ensure our tests sufficiently cover the core components of the system.
- **Linters**: The team will integrate language-specific linters, like Roslyn analyzers recommended by Microsoft, to ensure our code adheres to best practices and established coding standards.
- **Continuous Integration (CI)**: The team will try to use a continuous integration platform, such GitHub Actions. However, the unique demands of game development could make it difficult to achieve seamless

integration. Due to problems including lengthy build times, complex asset management, and dependence on graphical resources. Notwithstanding these possible obstacles, our goal is to set up a continuous integration pipeline that provides reliable feedback on code quality by automatically running tests whenever work is posted to the repository. It is important to note that this may not be entirely feasible.

2.7 Software Validation Plan

[If there is any external data that can be used for validation, you should point to it here. If there are no plans for validation, you should state that here. —SS]

[You might want to use review sessions with the stakeholder to check that the requirements document captures the right requirements. Maybe task based inspection? —SS]

[For those capstone teams with an external supervisor, the Rev 0 demo should be used as an opportunity to validate the requirements. You should plan on demonstrating your project to your supervisor shortly after the scheduled Rev 0 demo. The feedback from your supervisor will be very useful for improving your project. —SS]

[For teams without an external supervisor, user testing can serve the same purpose as a Rev 0 demo for the supervisor. —SS]

[This section might reference back to the SRS verification section. —SS]

Our software validation will involve confirming that the system meets user expectations and the requirements outlined in the SRS.

- **Stakeholder Review Sessions**: In order to make sure that requirements match user expectations and needs, we will arrange semi-regular review meetings with our supervisor and potential users, who are stakeholders. These meetings will be used to modify requirements as needed and will provide validation for newly developed components of the system.

- **Task-Based Inspection**: In order to demonstrate that the system performs as intended, the team will develop use-case scenarios and conduct task-based inspections to validate whether or not the system operates correctly.

- **Rev 0 Demo Feedback**: For the purpose of further refining the system and optimizing it prior to the final release, the team will collect feedback from our supervisor both before and after the Rev 0 demo.

These strategies are additions to the ones outlined in section 2.2.

3 System Tests

This section covers tests for functional and non-functional requirements to verify the game's adherence to its design specifications. The functional requirements focus on verifying the gameplay mechanics, score calculations, and user interface, while the non-functional requirements test aspects like performance, usability, and system compatibility.

3.1 Tests for Functional Requirements

Functional tests are organized into different areas based on gameplay features, score handling, and user interface components, as defined in the software requirements specification (SRS).

3.1.1 Gameplay and Rules

Testing here covers the core game mechanics, such as player vs player mode, score calculations, and dice roll simulation.

Test: PvP Support

- **Test ID:** Test-1
- **Control:** Automated
- **Initial State:** Game loaded to main menu.
- **Input:** Two players connect to a match.
- **Output:** Game begins with both players connected, and each takes turns as expected.
- **Test Case Derivation:** Confirms successful setup and turn-taking in online mode, satisfying R1.

- **How test will be performed:** Automated script connects two players in PvP mode and checks player statuses.

Test: Score Calculation

- **Test ID:** Test-2
- **Control:** Automated
- **Initial State:** Active game session with dice rolled.
- **Input:** Player achieves a Yahtzee (five of a kind).
- **Output:** Correct score added to the Yahtzee section.
- **Test Case Derivation:** Ensures that score calculations adhere to specified Yahtzee-like rules, satisfying R2.
- **How test will be performed:** Tester evaluates various combinations to ensure scoring accuracy.

Test: Dice Roll Physics

- **Test ID:** Test-3
- **Control:** Manual
- **Initial State:** Game session loaded.
- **Input:** Player rolls the dice.
- **Output:** 3D dice roll with randomized, realistic outcome.
- **Test Case Derivation:** Validates realistic physics for dice roll, ensuring that each outcome resembles real world physics, satisfying R3.
- **How test will be performed:** Tester initiates rolls and observes animation and outcomes.

Test: Simultaneous turn implementation

- **Test ID:** Test-4
- **Control:** Manual
- **Initial State:** Online game session loaded, Both players rolled once, one player confirmed choice of dice to reroll and is in waiting screen
- **Input:** Second player confirms choice of dice to reroll
- **Output:** Game rerolls chosen dice for both players, updates both players with the results of both rerolls, and allows both players to select new dice to reroll at the same time
- **Test Case Derivation:** Verifies that one player can't move to their next turn until both players are ready, thus enforcing the rules of Simultaneous turn implementation and satisfying requirement R6
- **How test will be performed:** Two testers in the same online session will end their turns in different orders to confirm the functionality of the implementation of Simultaneous turns.

3.1.2 User Interface and Settings

This section validates the interface, display, and customization options to ensure they meet user needs and game standards.

Test: User interface initiation

- **Test ID:** Test-5
- **Control:** Automatic
- **Initial State:** Game loaded to main menu
- **Input:** Player connects to a match
- **Output:** Game begins and a UI displaying important information, such as Player names, scores, number of rolls and time limits, is displayed.

- **Test Case Derivation:** confirms the Display of UI elements conveying important information to the User, thus satisfying requirement R8
- **How test will be performed:** Upon the beginning of a match, an automated script will Create a UI that displays important information to the user, and updates as the match goes on.

Test: Dice Selection

- **Test ID:** Test-6
- **Control:** Manual
- **Initial State:** Mid-game, with player allowed to roll again.
- **Input:** Player selects specific dice to keep and re-rolls the remaining.
- **Output:** Only unselected dice are re-rolled, while selected dice remain.
- **Test Case Derivation:** Validates dice selection mechanics as per game rules, satisfying R7.
- **How test will be performed:** Tester selects different dice combinations to confirm correct behavior.

Test: Game Presets

- **Test ID:** Test-7
- **Control:** Manual
- **Initial State:** Game loaded to game select menu
- **Input:** Player selects a preset game option
- **Output:** A new game session begins with the pre-made settings applied.
- **Test Case Derivation:** Confirms that each preset produces a unique variant of the game, Satisfying R10
- **How test will be performed:** Tester selects the pre-made options and runs the games to confirm functionality.

Test: Custom Game Settings Modification

- **Test ID:** Test-8
- **Control:** Manual
- **Initial State:** Custom Game settings menu open.
- **Input:** Player modifies settings (e.g., changes dice sides, scoring rules).
- **Output:** Changes are successfully applied to new game sessions.
- **Test Case Derivation:** Verifies that customizable game settings reflect unique variants, satisfying R9.
- **How test will be performed:** Tester modifies various settings and initiates games to confirm functionality.

3.2 Tests for Nonfunctional Requirements

Nonfunctional tests focus on measuring performance, responsiveness, reliability, and usability. These tests ensure the game performs well under different conditions and on various systems.

3.2.1 Performance and Usability

Test: Frame Rate Consistency

- **Test ID:** Test-9
- **Type:** Dynamic, Automated
- **Initial State:** Game in progress.
- **Input/Condition:** Standard gameplay, including dice rolls, UI interactions, and animations.
- **Output/Result:** Game maintains a frame rate of at least 30 FPS.
- **How test will be performed:** Frame rate monitored during gameplay on low and high-end systems to ensure stability.

Test: Input Responsiveness

- **Test ID:** Test-10
- **Type:** Dynamic, Automated
- **Initial State:** Game loaded.
- **Input:** Player performs multiple actions (e.g., roll dice, end turn).
- **Output:** System responds within 500 ms of input.
- **How test will be performed:** Response times recorded using automated tools, with any delays highlighted.

3.2.2 System Compatibility and Reliability

Test: System Compatibility

- **Test ID:** Test-11
- **Type:** Manual
- **Initial State:** Game installed on a Windows 10 system.
- **Input/Condition:** Launch and operate the game as per standard.
- **Output/Result:** Game loads and operates without compatibility issues.
- **How test will be performed:** Run game on Windows 10 (and MacOS - stretch goal) to confirm compatibility, satisfying NFR3 (and NFR10).

Test: Crash Resilience in Multiplayer

- **Test ID:** Test-12
- **Type:** Dynamic, Manual
- **Initial State:** Multiplayer game session in progress.
- **Input/Condition:** Two players connected and actively playing.

- **Output/Result:** Game remains stable, with crashes occurring less than 1% of the time.
- **How test will be performed:** Simulated load testing over numerous sessions, logging any crashes to validate NFR4.

3.3 Traceability Between Test Cases and Requirements

The following table provides traceability between the test cases and the requirements. Each test case is mapped to one or more requirements that it validates, ensuring coverage of our functional and non-functional requirements.

Table 3: Traceability Matrix for Test Cases and Requirements

Test Case ID	Requirement(s) Verified
Test-1	R1: Online player vs player mode
Test-2	R2: Score calculation using Yahtzee rules
Test-3	R3: Realistic 3D dice roll physics
Test-4	R6: Simultaneous turn implementation
Test-5	R8: User interface display
Test-6	R7: Dice selection functionality
Test-7	R10: Game Presets
Test-8	R9: Game settings modification
Test-9	NFR1: Frame rate consistency
Test-10	NFR5: Input responsiveness
Test-11	NFR3: Windows 10 compatibility, NFR10: MacOS support
Test-12	NFR4: Multiplayer stability

4 Unit Test Description

[This section should not be filled in until after the MIS (detailed design document) has been completed. —SS]

[Reference your MIS (detailed design document) and explain your overall philosophy for test case selection. —SS]

[To save space and time, it may be an option to provide less detail in this section. For the unit tests you can potentially layout your testing strategy here. That is, you can explain how tests will be selected for each module.

For instance, your test building approach could be test cases for each access program, including one test for normal behaviour and as many tests as needed for edge cases. Rather than create the details of the input and output here, you could point to the unit testing code. For this to work, you code needs to be well-documented, with meaningful names for all of the tests. —SS]

4.1 Unit Testing Scope

[What modules are outside of the scope. If there are modules that are developed by someone else, then you would say here if you aren't planning on verifying them. There may also be modules that are part of your software, but have a lower priority for verification than others. If this is the case, explain your rationale for the ranking of module importance. —SS]

4.2 Tests for Functional Requirements

[Most of the verification will be through automated unit testing. If appropriate specific modules can be verified by a non-testing based technique. That can also be documented in this section. —SS]

4.2.1 Module 1

[Include a blurb here to explain why the subsections below cover the module. References to the MIS would be good. You will want tests from a black box perspective and from a white box perspective. Explain to the reader how the tests were selected. —SS]

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

2. test-id2

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

3. ...

4.2.2 Module 2

...

4.3 Tests for Nonfunctional Requirements

[If there is a module that needs to be independently assessed for performance, those test cases can go here. In some projects, planning for nonfunctional tests of units will not be that relevant. —SS]

[These tests may involve collecting performance data from previously mentioned functional tests. —SS]

4.3.1 Module ?

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

4.3.2 Module ?

...

4.4 Traceability Between Test Cases and Modules

[Provide evidence that all of the modules have been considered. —SS]

References

5 Appendix

This is where you can place additional information.

5.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

5.2 Usability Survey Questions?

[This is a section that would be appropriate for some projects. —SS]

Appendix — Reflection

[This section is not required for CAS 741 —SS]

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning.

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?
    - We were able to use past year examples to set up our document and understand what contents to add to each section - Hemraj B
2. What pain points did you experience during this deliverable, and how did you resolve them?
    - Sections 3 and 4, were confusing in terms of us thinking that both essentially meant the same thing. We resolved this by looking at the previous year's examples and were able to figure out the differences between the two sections - Hemraj B
3. What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? Examples of possible knowledge and skills include dynamic testing knowledge, static testing knowledge, specific tool usage, Valgrind etc. You should look to identify at least one item for each team member.
    - We will need to familiarize ourselves with testing methods that were mentioned in our document such as Coverlet and FxCop - Hemraj B

4. For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?

    - We will acquire knowledge of Coverlet and FxCop through watching educational videos on YouTube and we will create trial projects to practice the software and familiarise ourselves with the tools available on each software - Hemraj B