System Verification and Validation Plan for Sandlot

Team 29
Nicholas Fabugais-Inaba
Casra Ghazanfari
Alex Verity
Jung Woo Lee

November 4, 2024

Revision History

Date	Version	Notes
Oct. 28, 2024	1.0	TA Feedback
Nov. 1, 2024	1.1	Rev0

Contents

1 Symbols, Abbreviations, and Acronyms

symbol	description
SRS	Software Requirements Specification
GSA	Graduate Students Association
VnV	Verification and Validation
CI	Continuous Integration
T	Test

This document will outline the various VnV plans for the Sandlot project including the system tests and unit tests that will be used to validate the requirements created in the SRS document.

2 General Information

2.1 Summary

The software being tested is the Sandlot project. Sandlot is a scheduling and management platform for the McMaster GSA softball league. Users of the system will include players, captains, commissioners, and other general users that will not need to make an account to access the system. Sandlot is intended to be an upgrade to the current platform that is outdated and lacks maintainability. This project will build off the current platform's existing features such as game scheduling, viewing of the scoring and standings, and team creation. Sandlot will also add new features including account creation and commissioner specific permissions like league-wide alerts.

2.2 Objectives

The main objectives intended to be accomplished are to build confidence in the software correctness, demonstrate the requirements created for this project are correctly implemented, and demonstrate adequate usability for functionalities of the system. When demonstrating the correct implementations of requirements for the project, these requirements are referring to the ones outlined in both the SRS and Hazard Analysis documents.

Objectives that are out of scope due to the limitations in our resources for verification and validation include the assumption that the database and the web server the platform utilizes and the platform runs on, respectively, have already been verified by its implementation teams.

2.3 Challenge Level and Extras

The challenge level of the project is general. The extras that are being used are user documentation and a code walkthrough. The challenge level and both extras have been approved by an instructor.

2.4 Relevant Documentation

References

[1] Software Requirements Specification Document (2024) SRS.pdf

[2] Development Plan Document (2024) DevelopmentPlan.pdf

[3] Module Guide Document (2024) MG.pdf

[4] Module Interface Specification Document (2024) MIS.pdf

The SRS document has requirements for our project that must be verified, it is the most complete list of functional and non-functional requirements that our solution needs to achieve.

The Development Plan Document contains information about our plans on how to use automated testing and verification tools for the project. Specifically, it discusses linting tools, unit testing frameworks, code coverage measuring tools, performance measuring tools, and our plans for CI. All of which are important tools used to help us verify and validate our tests.

The MG and MIS documents also refer heavily to the requirements being tested in our VnV plan.

3 Plan

In this section our plan for verifying all important documents will be recorded as well as our plan for verifying the software itself. Documents to be verified include the SRS, MIS and MG design documents and the VnV itself.

3.1 Verification and Validation Team

1. Casra: Automation Tester
   Will focus on automation testing using GitHub Actions.

2. Jung Woo: Non-Functional Tester
   Will focus on non-functional tests listed in section 4.2.

3. Alex: Functional Tester
   Will focus on functional tests listed in section 4.1.

4. Nicholas: Survey Tester
   Will be in charge of performing surveys needed for any non-functional tests.

5. Dr. Jake Nease: Supervisor
   Will help give feedback on all functionality of the solution and gather stakeholders to help test the solution.

3.2 SRS Verification Plan

A meeting with the team and supervisor will be held to verify the SRS document covers all of the necessary requirements desired for the system. Each requirement outlined in the SRS will be reviewed by the team and supervisor to ensure the requirement appropriately addresses the requested functionality.

The SRS document will also be reviewed by another capstone group who will give at least six points of feedback on our SRS document. Our team will review the feedback and make any changes needed.

3.3 Design Verification Plan

A meeting with the team and supervisor will be held to review design documents created for the Sandlot project. Both parties will then verify anything outlined in the design documents are correctly implemented in the system.

An additional meeting with the team will be held to verify any and all reviews created by classmates are either implemented in the system or appropriately addressed by a team member with a justification for why a suggestion shall not be implemented.

3.4 Verification and Validation Plan Verification Plan

The verification and validation plan will be reviewed by another capstone group who will give at least six points of feedback on our VnV document. Our team will review the feedback and make any changes needed.

During the writing of the verification and validation plan, any questions or concerns for the superviso will be recorded and asked during our next scheduled supervisor meeting.

During testing we will use mutation testing to verify test case coverage, and any holes found in our coverage will be patched by adding more test cases.

3.5 Implementation Verification Plan

Code reviews with the team will be held for all functionalities of the system, utilizing the system tests and unit tests outlined in the VnV plan. The team will attempt to identify any errors within the system and correct any faults that may occur.

Furthermore, a code walkthrough and user documentation will be created by the team and reviewed by the supervisor. Both the code walkthrough and user documentation must highlight all functionalities of the system and how they are implemented. The supervisor will then verify if the code walkthrough and user documentation adequately supply the necessary information for an admin or developer to understand the full solution. Their understanding should allow them to sufficiently maintain the system and add any new functionalities as required.

3.6 Automated Testing and Verification Tools

This was discussed in our development plan document (2) in the expected technology section.

3.7 Software Validation Plan

The system will be provided to the supervisor and external testers, gathered by the supervisor, to enter given inputs into the system. The given inputs shall produce expected outputs according to the system tests created from the SRS. For specific system tests, a usability survey will be provided along with the test that must be answered by the individual conducting the test.

4 System Tests

This section will list all test cases for both functional and non-functional requirements, split into distinct areas of testing. The goal of the test cases is to have full coverage over all possible errors. This section will likely be added to in future, as holes in coverage are found and new test cases are made.

4.1 Tests for Functional Requirements

This section covers tests verifying all functional requirements in our SRS document (1). Areas of testing include scheduling, accounts, team structure, scoring/standings and alerts. These areas cover all major functionality of our solution.

4.1.1 Scheduling

This section includes all functional tests related to viewing, creating and modifying the league schedule. The schedule includes all future and past matches created using each team's availability at the start of the season and can be modified with the reschedule feature.

1. test-FR8

   Control: Manual

   Initial State: The system has provided the option for captains to enter their team availability data and is ready to take in the user's input.

   Input: Non-conflicting team availability data.

   Output: Captain inputted team availability data is stored in the system.

   Test Case Derivation: The team availability data inputted by the captain has been determined to not conflict with other availability data already stored in the system.

   How test will be performed: The tester will select the option to enter in team availability data and be provided non-conflicting team availability data that they will input into the system. They will then observe if the system successfully accepts the inputted team availability data and if any errors occur within the system.

2. test-FR9

   Control: Automatic

   Initial State: The current seasons due date for inputting availability has been reached.

   Input: The current season's inputted team availability data.

   Output: A season schedule is generated by the system and a visual representation of the schedule is generated and displayed on the website.

   Test Case Derivation: Players should be able to quickly know when and where they're playing their games shortly after they've submitted their availability.

   How test will be performed: A wide set of availability data sets will be submitted to the schedule generation feature, with the expected results compared to the results given by the program.

3. test-FR10

   Control: Manual

   Initial State: There are two captains whose teams are scheduled to play a game in the future.

   Input: One captain submits a reschedule request.

   Output: The other captain receives a reschedule request.

   Test Case Derivation: A captain should receive a reschedule request if another captain submits a request on a game both captains will be playing.

   How test will be performed: The tester will check if a reschedule request is successfully sent when a captain requests a reschedule. This will be checked for at least 3 different dates and times.

4. test-FR11

   Control: Manual

   Initial State: A reschedule request has been sent to a captain.

Input: A captain accepts or denies a reschedule request.

Output: If accepted, the game is rescheduled to that time. If denied, the captain who made the request is notified of the denial.

Test Case Derivation: A captain should be able to accept or deny a reschedule request.

How test will be performed: The tester will accept or deny multiple reschedule requests on different dates and times and verify the correct output is made by the system.

5. test-FR12

   Control: Manual

   Initial State: The system has received a captain's reschedule request and is ready to notify them about the outcome of their request.

   Input: Captain reschedule request.

   Output: A notification is sent by the system to the captain about the status of their reschedule request.

   Test Case Derivation: The notification about the status of a captain's reschedule request is immediately sent to the captain once the status is confirmed to be either accepted or denied by the system.

   How test will be performed: The tester will be provided both a valid and invalid captain reschedule request to submit into the system and wait to observe the notification sent by the system of the status for the reschedule request. They will then observe if the system has both accepted and denied the submitted reschedule requests and if any errors occur within the system.

6. test-FR18

   Control: Manual

   Initial State: The system has created a league schedule and is ready to accept new schedule data.

   Input: New schedule data.

Output: The updated leage schedule according to the new schedule data inputted.

Test Case Derivation: The leage schedule is immediately updated and shows the new changes once new schedule data is inputted into the system.

How test will be performed: The tester will be provided admin level permissions and sample schedule data to submit into the system and wait to observe the updated league schedule by the system once the new schedule data is received. They will then observe if the system has displayed the correct updated league schedule and if any errors had occured within the system.

7. test-FR20

   Control: Manual

   Initial State: The system has created a season schedule.

   Input: Season schedule.

   Output: Schedule of all games of all teams during the season in a calendar.

   Test Case Derivation: The overall season schedule should display all games in a calendar.

   How test will be performed: The season schedule should resemble a calendar and display all games of the season.

8. test-FR21

   Control: Manual

   Initial State: The system has created a schedule.

   Input: User navigating to a team's schedule section.

   Output: The system will display all games of the specified team's schedule.

   Test Case Derivation: Each team should have a schedule that lists all of that team's games. This should be accessible by users who navigate to this section of the system.

How test will be performed: Each team's schedule section will be navigated to and compared with the full league schedule. All games on the main schedule that include the specified team should be on the team's schedule. There should not exist any games on the team's schedule where the specified team is not playing.

9. test-FR22

   Control: Manual

   Initial State: The system has created a schedule and is ready to display upcoming games from an accepted time interval.

   Input: Time interval.

   Output: The upcoming games in a schedule according to the time interval inputted.

   Test Case Derivation: The schedule shows all upcoming games within the inputted time interval that is received by the system.

   How test will be performed: The tester will be provided a sample time interval to input into the system and wait to observe all of the upcoming games from the displayed schedule. They will then observe if the system has displayed the correct schedule based on the inputted time interval and if any errors had occured within the system.

4.1.2 Accounts

This section includes all functional tests related to the creation, use and modification of accounts.

1. test-FR1-1

   Control: Manual

   Initial State: System is open on user's browser.

   Input: User requests to display the season schedule.

   Output: Season schedule is displayed.

   Test Case Derivation: The season schedule should be displayed to all users regardless of their access level.

How test will be performed: Users will attempt to display the season schedule.

2. test-FR1-2

   Control: Manual

   Initial State: System is open on user's browser.

   Input: User requests to display the standings.

   Output: Standings are displayed.

   Test Case Derivation: Standings should be displayed to all users regardless of their access level.

   How test will be performed: Users will attempt to display the standings.

3. test-FR3-1

   Control: Manual

   Initial State: The system is not logged in to an account.

   Input: User navigates to create an account and enters valid account creation data.

   Output: The system adds an account to the database and logs in to the new account.

   Test Case Derivation: If valid account information is given a new account should be created.

   How test will be performed: Multiple accounts will be added to the system with differing valid account data covering all input fields.

4. test-FR3-2

   Control: Manual

   Initial State: The system is not logged in to an account.

   Input: User navigates to create an account and enters invalid account creation data.

   Output: The system does not create a new account and the user is informed of which data is invalid.

Test Case Derivation: If invalid account information is given a new account should not be created and the user should be notified of which data is invalid.

How test will be performed: Multiple attempts to create accounts with differing invalid account data will be made, with invalid data each attempt covering different input fields.

5. test-FR4

   Control: Manual

   Initial State: The system is set up and ready to take in the user's input.

   Input: Valid account information.

   Output: User inputted account information has replaced the previously displayed account information.

   Test Case Derivation: The account information inputted by the user has already been determined to be valid and should not cause the system to run into any errors. The user inputted account information, although should be correct, is not required by the system to be correct to change the user's previous account information.

   How test will be performed: The tester will be provided valid account information that they will input into the system and observe if the system successfully accepts the inputted account information. The tester will also observe if the previously existing account information has been changed to the information that had been entered at the start of the test. At any point during the test, the tester will also observe if any errors occur within the system.

6. test-FR5-1

   Control: Manual

   Initial State: User is logged into an account.

   Input: User requests to delete their account and provides valid login information for that account.

   Output: The user is logged out, and their account is deleted.

Test Case Derivation: If valid login information is submitted then the account should be deleted if the user requests to do so because if the user wishes to leave the league they should be able to delete their information from the system.

How test will be performed: Users of all account types will attempt to delete their account using valid login information.

7. test-FR5-2

   Control: Manual

   Initial State: User is logged into an account.

   Input: User requests to delete their account and provides invalid login information for that account.

   Output: The user is not logged out, their account is not deleted, and the user is told why their request was unsuccessful.

   Test Case Derivation: If invalid login information is submitted then the account should not be deleted because accounts should only be able to be deleted if the security protections in place are satisfied. Additionally, the user should be informed about why the request failed because if the user requesting account deletion is the proper owner of the account but is facing issues they should be provided information to help them troubleshoot the issues.

   How test will be performed: Users of all account types will attempt to delete their account using invalid login information.

8. test-FR16-1

   Control: Manual

   Initial State: The system is not logged in to an account.

   Input: User navigates to the log in section and enters valid login data for an account that has already been made.

   Output: The system logs in to the valid account.

   Test Case Derivation: If valid account information is given when logging in the system should log in as that account.

How test will be performed: Multiple accounts will be logged into each with different valid account information and permission levels.

9. test-FR16-2

   Control: Manual

   Initial State: The system is not logged in to an account.

   Input: User navigates to the log in section and enters invalid login data for an account login.

   Output: The system warns the user the data used is invalid.

   Test Case Derivation: If invalid account information is given when logging in the system should not log in to any account and warn the user the data isn't valid.

   How test will be performed: Multiple invalid logins will be attempted each with different invalid account information.

10. test-FR19

    Control: Manual

    Initial State: The User is logged into a commissioner account.

    Input: The user requests to assign captain level permissions to a certain user.

    Output: The system assigns captain level permissions to the user.

    Test Case Derivation: commissioners should be able to assign captain level permissions to users because they control which players are captains.

    How test will be performed: Users logged into commissioner accounts will attempt to assign captain level permissions to a player level account.

4.1.3 Team Structure

This section includes all functional tests related to teams and team information. This covers team creation, users joining teams, and team information being modified.

1. test-FR2-1

   Control: Manual

   Initial State: Captain user is logged in and subject team has not been created in system.

   Input: Captain user navigates to team creation and inputs valid team information then submits.

   Output: Submitted team is added to the database.

   Test Case Derivation: If valid team information is submitted, team should be created and information reflected in the database.

   How test will be performed: Captain accounts will create multiple valid teams with differing data that covers all input fields.

2. test-FR2-2

   Control: Manual

   Initial State: Captain user is logged in.

   Input: Captain user navigates to team creation and inputs invalid team information then submits.

   Output: No team is added to the database and user is given informative feedback as to why team submission failed.

   Test Case Derivation: If invalid team information is submitted, team should not be created and user should be told the reason as to why.

   How test will be performed: Captain accounts will attempt to create multiple invalid teams with differing data that covers all input fields.

3. test-FR7

   Control: Automatic

   Initial State: The system is logged into a commissioner level account.

   Input: New team data to replace a current team's data.

   Output: If data is valid, system should replace the old team data with the new team data. If the data is invalid, the system should not change the old team data and inform the commissioner the data is invalid.

Test Case Derivation: A commissioner level account should be able to change any team's data, including player list and scores.

How test will be performed: A set of valid and invalid test cases will be submitted to the commissioner's replace team data feature, with the expected results compared to the results given by the program.

4. test-FR13-1

   Control: Manual

   Initial State: The User is logged into a captain account.

   Input: The user requests to update the team information of the team they are the captain of.

   Output: The system provides the user with a menu to change the team's information and will save any changes made by the user.

   Test Case Derivation: Captains of teams should be able to update the team information of their team because certain team information may have changed since the team was created.

   How test will be performed: Users logged into captain accounts will attempt to request to update the team information of the team they are the captain of.

5. test-FR13-2

   Control: Manual

   Initial State: The User is logged into a captain account.

   Input: The user requests to update the team information of a team they are not the captain of.

   Output: The system informs the user that they are not the captain of this team and therefore cannot update this team's information.

   Test Case Derivation: The team information of a team should only be able to be updated by the captain of that team, not captains of other teams.

   How test will be performed: Users logged into captain accounts will attempt to request to update the team information of a team they are not the captain of.

6. test-FR14-1

   Control: Manual

   Initial State: The User is logged into a captain account.

   Input: The user requests to submit the score for a game that has been completed.

   Output: The system provides the user with a menu to submit the game's score and will save the score inputted by the user.

   Test Case Derivation: Captains of teams should be able to submit the score for a game that has been completed because they're responsible for the score.

   How test will be performed: Users logged into captain accounts will attempt to submit a score for a game that has been marked as completed by the system.

7. test-FR14-2

   Control: Manual

   Initial State: The User is logged into a captain account.

   Input: The user requests to submit the score for a game that has not been completed.

   Output: The system informs the user that they cannot submit a score for this game because it has not been completed.

   Test Case Derivation: Captains of teams should not be able to submit the score for a game that has not been completed because the final score for that game has not been determined yet.

   How test will be performed: Users logged into captain accounts will attempt to submit a score for a game that has not been marked as completed by the system.

8. test-FR14-3

   Control: Manual

   Initial State: The User is logged into a player account.

Input: The user requests to submit the score for a game that has been completed.

Output: The system informs the user that they are not the captain of this team and therefore cannot update the score for this game.

Test Case Derivation: regular players of a team should not be able to submit the score for a game because that is not their responsibility.

How test will be performed: Users logged into player accounts will attempt to submit a score for a game that has been marked as completed by the system.

9. test-FR15

   Control: Manual

   Initial State: The system is logged into a player level account.

   Input: Join team interaction.

   Output: Player is added to the team. This is reflected in team composition and alerts sent to that team.

   Test Case Derivation: A player who has joined a team should be shown to be a member of that team in the system.

   How test will be performed: Multiple player accounts will attempt to join different teams. Team compositions will be inspected to see if the player is shown to be a member.

10. test-FR23

    Control: Manual

    Initial State: At least one team has been created by a captain.

    Input: A captain level user is made the associated captain of a team that already has an associated captain.

    Output: Feedback to the user that the team already has a captain in charge.

    Test Case Derivation: Only one captain is allowed to be associated with a team due to league rules.

How test will be performed: A team with an associated captain will attempt to have another captain be associated with it. It is important to note that a captain playing for a a different team than the one they are in charge of does not count as being associated, they are just playing for the team.

4.1.4 Scoring/Standings

This section includes all functional tests related to the scoring and league standings. It mainly focuses on captains recording the scores of matches their team plays.

1. test-FR15

   Control: Manual

   Initial State: The system will have a game in the schedule that will be played in the future. The system is logged in as a captain.

   Input: A score submitted for a game.

   Output: No output.

   Test Case Derivation: A score should not be entered by a captain until a game is played.

   How test will be performed: A captain level account will attempt to submit a score for a game that has not been played yet. It should not modify the standings.

4.1.5 Alerts

This section includes all functional tests related to the creation of alerts. Alerts can be created by commissioners and send information to any specified group of users of the solution.

1. test-FR6-1

   Control:

   Initial State: The system is logged into a commissioner level account.

   Input: Valid alert data and chosen target user(s) entered.

Output: Alerts are stored in the system. Alerts are viewable by target users.

Test Case Derivation: Alerts sent to target user should be able to be seen by those target users.

How test will be performed: Different commissioner level accounts input various valid alert data and submit. Target user accounts should be checked if the alerts are received.

2. test-FR6-2

   Control:

   Initial State: The system is logged into a commissioner level account.

   Input: Invalid alert data or invalid chosen target user(s) entered.

   Output: User is given informative feedback as to why alert failed to submit.

   Test Case Derivation: Invalid alerts should not be sent and user should be told as to why they could not be sent.

   How test will be performed: Commissioner level account inputs various invalid alert data covering all input fields.

4.2 Tests for Nonfunctional Requirements

This section covers tests verifying all non-functional requirements in our SRS document (1). Most prominent areas of testing include usability, performance, maintainability, security and cultural.

4.2.1 Look and Feel Requirements

1. test-AP1

   Type: Non-Functional, Dynamic, Manual

   Initial State: The solution is open on the user's browser.

   Input: The user navigates through the website.

Output: All input elements displayed have a minimum size of 44x44 pixels, maintain and maintain at least a 3:1 contrast ratio with the background.

How test will be performed: Each possible user input element will be checked by the user to have a minimum size of 44x44 pixels, maintain and maintain at least a 3:1 contrast ratio with the background.

2. test-AP2

   Type: Non-Functional, Dynamic, Manual

   Initial State: The system is logged in.

   Input: Inputs associated with each different type of input field in the solution.

   Output: Feedback from each input field.

   How test will be performed: Each possible type of input will be used by the tester and they will monitor the result for feedback. Each input field will be tested with valid and invalid data if applicable, and invalid data should receive feedback that lets the user know the input was invalid.

3. test-AP3

   Type: Non-Functional, Dynamic, Manual

   Initial State: The solution is opened on a user's web browser.

   Input/Condition: The user will be asked if the images made for/by Sandlot are viewed at a high quality containing no pixelations or blurring at their displayed size.

   Output/Result: The supervisor will state that Sandlot's images are displayed at a high quality.

   How test will be performed: The supervisor will be provided the solution and a set of sample inputs for Sandlot. They will then enter in the sample inputs and observe the generated outputs from the system. After their observations, the supervisor will be given a usability survey to fill out that is located in section 6.2 of this VnV plan document.

4. test-AP4

   Type: Non-Functional, Dynamic, Manual

   Initial State: The solution is opened on a user's web browser.

   Input/Condition: The user will be asked if the navigation of Sandlot is straightforward and if menus and links easily accessible and readable.

   Output/Result: The user will state that Sandlot's navigation is straightforward.

   How test will be performed: The user will be provided the solution. They will then navigate throughout the system and observe its simplicity. After their observations, the supervisor will be given a usability survey to fill out that is located in section 6.2 of this VnV plan document.

5. test-STY1

   Type: Non-Functional, Dynamic, Manual

   Initial State: The system is running and viewable.

   Input/Condition: User will be asked if there are inconsistent colours, fonts, or buttons across the interface of the system.

   Output/Result: Supervisor will state that the style is consistent.

   How test will be performed: Reviewer will navigate through all parts of the interface and inspect text, colours, and buttons. Any inconsistent interface elements will be recorded.

4.2.2 Usability and Humanity Requirements

1. test-EU1

   Type: Non-Functional, Dynamic, Manual

   Initial State: The system is not logged in.

   Input: Navigation inputs leading to the full season schedule.

   Output: The season schedule displayed on the screen.

How test will be performed: At least 5 testers unfamiliar to the system will attempt to navigate to the season schedule. Their number of clicks used and time taken to get to the season schedule will be recorded. The test passes if on average testers take less than 2 clicks and less than one minutes to find the schedule.

2. test-EU2

   Type: Non-Functional, Dynamic, Manual

   Initial State: User is located on the system's login page and the system is ready for the user's inputs.

   Input/Condition: Misinputted login information.

   Output/Result: The system will provide a warning to the user for login information that does not exist or does not match any database stored login information.

   How test will be performed: The tester will be provided login information that does not currently exist in the database and they will input the provided information into the system. The tester will observe the output or any errors that may occur in the system.

3. test-EU3

   Type: Non-Functional, Dynamic, Manual

   Initial State: User is logged into a captain account.

   Input/Condition: User inputs availability data which causes a schedule conflict.

   Output/Result: The system will provide a warning to the user informing them that the availability data they've inputted has caused on conflicts with another team's availability data that the scheduler cannot resolve.

   How test will be performed: The user will be provided a captain account with a set of availability data. The system will have stored some set of existing availability data that will cause an unresolvable conflict with the availability data of the user. The user will input the provided

availability data to the system and observe the output or any errors that may occur in the system.

4. test-LR1

   Type: Non-Functional, Dynamic, Manual

   Initial State: The system is logged in to a commissioner level account.

   Input/Condition: Inputs for all features available to commissioner level accounts.

   Output/Result: The results of using each commissioner level account feature.

   How test will be performed: A tester will be given a list of tasks to complete that involve commissioner specific actions including sending alerts and editing team composition. The tester will also be given the user manual. If the user can complete all tasks in less than one hour the test is successful.

5. test-LR2

   Type: Non-Functional, Dynamic, Manual

   Initial State: The system is logged in to a captain level account.

   Input: Inputs for all features available to captain level accounts.

   Output: The results of using each captain level account feature.

   How test will be performed: A tester will be given a list of tasks to complete including creating a team, modifying team data and submitting a score. The tester will also be given the user manual. If the user can complete all tasks in less than one hour the test is successful.

6. test-LR3

   Type: Non-Functional, Dynamic, Manual

   Initial State: The solution is opened on a user's web browser.

   Input/Condition: A new user will be asked to navigate to the season schedule on their first time interacting with the solution.

Output/Result: A new user is able to successfully navigate to the season schedule on their first time interacting with the solution.

How test will be performed: A new user of the system will be provided the solution and will be asked to navigate and view the season schedule.

7. test-UP1

   Type: Non-Functional, Dynamic, Manual

   Initial State: The system is running.

   Input/Condition: User will be asked to navigate through system to selected areas of text for them to read.

   Output/Result: Users will be able to understand terminology used in the selected tests without difficulty. If 90 percent of users do understand, the test is considered a success.

   How test will be performed: A set of testers will read selected text in the system and record whether they generally do or do not understand the terminology used.

8. test-AC1

   Type: Non-Functional, Dynamic, Manual

   Initial State: The solution is open on the user's browser.

   Input: The user navigates through the website.

   Output: All body text displayed have a minimum font size of 16 pixels, and a line length between 45 and 75 characters.

   How test will be performed: All possible body text will be checked by the user to have a minimum font size of 16 pixels, and a line length between 45 and 75 characters.

9. test-AC2

   Type: Non-Functional, Dynamic, Manual

   Initial State: The system is logged in to a commissioner level account.

   Input: Navigation to each page of the system.

Output: The colours on screen of each page of the system.

How test will be performed: Each page of the system will be viewed, and any adjacent colours will be confirmed to have a ratio of at least 4.5:1. Particular attention will be given to input fields and critical information such as schedules.

10. test-AC3

    Type: Non-Functional, Dynamic, Manual

    Initial State: The solution is opened on a user's web browser and a commissioner's web browser.

    Input/Condition: A user will be sent an alert created by a commissioner and will be asked if the alert is visible and readable.

    Output/Result: A user is able to successfully see the alert sent by a commissioner and the alert is readable and clear enough for the user to understand.

    How test will be performed: The supervisor and a tester will be provided the solution and will be asked to view/send an alert. Once the alert is sent/received, the recipient will be asked to observe the alert. After their observations, the supervisor will be given a usability survey to fill out that is located in section 6.2 of this VnV plan document.

4.2.3 Performance Requirements

1. test-SL1

    Type: Non-Functional, Dynamic, Manual

    Initial State: The solution is not loaded, internet connection is stable.

    Input/Condition: The system is started.

    Output/Result: Sandlot loads and displays the homepage within 3 seconds.

    How test will be performed: The system will be timed from not loaded to a fully loaded homepage. This will be recorded 10 times. If the average load time is less than 3 seconds the test is successful.

2. test-CR1

   Type: Non-Functional, Dynamic, Manual

   Initial State: The solution is opened on a user's web browser.

   Input/Condition: Sandlot should have 60 teams and 1500 players stored in the system at once.

   Output/Result: Sandlot should continue to successfully function with no faults when 60 teams have 25 players each stored in the system.

   How test will be performed: The system will be provided sample inputs for 60 teams and 25 players on each team. The tester will then observe the system as they navigate the solution and utilize functionalities such as rescheduling games, and viewing the standings and season schedule. After their observations, the tester will be given a usability survey to fill out that is located in section 6.2 of this VnV plan document.

3. test-CR2

   Type: Non-Functional, Dynamic, Manual

   Initial State: The system is online.

   Input/Condition: Sandlot will have 500 users using the system at the same time.

   Output/Result: Sandlot should continue to successfully function with no faults when 500 users are using the system at once.

   How test will be performed: 500 users will be mimicked using a solution like JMeter. All functions on the system would be run through to make sure no faults occur.

4.2.4 Operational and Environmental Requirements

1. test-IAS1

   Type: Non-Functional, Dynamic, Manual

   Initial State: Sandlot is running.

Input/Condition: User will attempt to access system from multiple browsers with each of the latest four releases of the product.

Output/Result: User is able to access and use system from multiple browsers for the latest four releases of the product.

How test will be performed: Tester will attempt to use product on the five most popular web browsers for each of the latest four releases of the product. The user should be able to use the product with no issues on each browser for each version of the product.

2. test-RR1

No test needed for requirement.

4.2.5 Maintainability and Support Requirements

1. test-MR1

Type: Non-Functional, Dynamic, Manual

Initial State: The solution is opened on a user's web browser.

Input/Condition: A tester will be asked to start a new season within a time constraint of one hour.

Output/Result: A new season is successfully started within one hour of the solution being initially opened in the tester's web browser.

How test will be performed: A tester will be provided the solution and asked to start a new season by following a set of instructions. A timer will begin at the same time the solution is opened on the tester's web browser, in which, the tester should be able to start a new season before one hour has passed.

4.2.6 Security Requirements

1. test-AS1-1

Control: Manual

Initial State: System is not logged in to an account.

Input: User requests to display the season schedule.

Output: Season schedule is displayed.

How test will be performed: Users without accounts will attempt to display the season schedule.

2. test-AS1-2

   Control: Manual

   Initial State: System is not logged in to an account.

   Input: User requests to display the standings.

   Output: Standings are displayed.

   How test will be performed: Users without accounts will attempt to display the standings.

3. test-AS2-1

   Type: Non-Functional, Dynamic, Manual

   Initial State: The system is logged in to a player level account.

   Input/Condition: Player attempts to access the contact information of their team captain.

   Output/Result: Captain's contact information is displayed.

   How test will be performed: Various player level accounts from various teams will attempt to access the contact information of their respective team captains. Players should be able to access the information.

4. test-AS2-2

   Type: Non-Functional, Dynamic, Manual

   Initial State: The system is logged in to a player level account.

   Input/Condition: Player attempts to access the contact information of a team captain not in their team.

   Output/Result: Captain's contact information is not displayed.

How test will be performed: Various player level accounts from various teams will attempt to access the contact information team captains who are not in their respective teams. Players should not be able to access the information.

5. test-AS3-1

   Type: Non-Functional, Dynamic, Manual

   Initial State: The system is logged in to a captain level account.

   Input/Condition: The captain will attempt to access the contact information of different players on their team and other captains in the league.

   Output/Result: The system will display their contact information.

   How test will be performed: The user will attempt to access contact information of multiple players on their team and other captains in the league. They should be able to access it.

6. test-AS3-2

   Type: Non-Functional, Dynamic, Manual

   Initial State: The system is logged in to a captain level account.

   Input/Condition: The captain will attempt to access the contact information of different players that are not on their team.

   Output/Result: The system will not display their contact information.

   How test will be performed: The user will attempt to access contact information of multiple players who are not on their team. They should not be able to access it.

7. test-AS4

   Type: Non-Functional, Dynamic, Manual

   Initial State: The system is logged in to a commissioner level account.

   Input/Condition: The commissioner will attempt to access the contact information of many different players and captains on different teams.

Output/Result: The system will display their contact information.

How test will be performed: The user will attempt to access contact information of multiple players and captains on different teams. They should be able to access it.

8. test-AS5

   Type: Non-Functional, Dynamic, Manual

   Initial State: The solution is opened on a user's web browser.

   Input/Condition: A user's account is provided permissions that are different from the the account's current permissions.

   Output/Result: The system provides a warning to the user changing an account's permissions before the permissions are successfully updated.

   How test will be performed: A tester will be provided an account to change the permissions for and the account itself. They will then be asked to update the account that requires an update to its permissions and observe the warning given by the system or if any errors had occurred. The tester will then be asked to log in to the account that recently had its permissions change and try out new functionalities that should have been provided with the permission update or if any errors had occurred.

9. test-AS6-1

   Type: Non-Functional, Dynamic, Manual

   Initial State: The system is not logged in to an account.

   Input: User navigates to the log in section and enters valid login data for an account that has already been made.

   Output: The system logs in to the valid account.

   Test Case Derivation: If valid account information is given when logging in the system should log in as that account.

   How test will be performed: Multiple accounts will be logged into each with different valid account information and permission levels.

10. test-AS6-2

    Type: Non-Functional, Dynamic, Manual

    Initial State: The system is not logged in to an account.

    Input: User navigates to the log in section and enters invalid login data for an account login.

    Output: The system warns the user the data used is invalid.

    Test Case Derivation: If invalid account information is given when logging in the system should not log in to any account and warn the user the data isn't valid.

    How test will be performed: Multiple invalid logins will be attempted each with different invalid account information.

11. test-AS7

    Type: Non-Functional, Dynamic, Manual

    Initial State: The system is logged in as a captain of a team.

    Input/Condition: Captain user attempts to join a team that they are not a part of.

    Output/Result: The captain user will be able to join the team.

    How test will be performed: A captain of a team will attempt to join a team that is not their own. It should be checked if the captain user is successfully added to the team list.

12. test-AS8

    Type: Non-Functional, Dynamic, Manual

    Initial State: The system is logged in as a captain of a team and is also a player on another team.

    Input/Condition: The captain attempts to change team information and submit scores.

    Output/Result: The system should not make any changes.

How test will be performed: A captain who is a player on a team they are not in charge of will try to change team information and submit scores for the team they are not in charge of. If they cannot do these actions, the test succeeds.

13. test-IG1-1

    Type: Non-Functional, Dynamic, Manual

    Initial State: The system has received scheduling data.

    Input/Condition: The system is instructed to create the schedule with non-conflicting scheduling data.

    Output: The system creates a schedule without conflicting scheduling data.

    How test will be performed: Sample scheduling data will be provided to the system. The test succeeds if a schedule is created without conflicting scheduling data.

14. test-IG1-2

    Type: Non-Functional, Dynamic, Manual

    Initial State: The system has scheduling data and awaits any reschedule requests.

    Input/Condition: The system is sent a reschedule request.

    Output: The system accepts or denies a reschedule request and either changes the scheduling data or remains the same with no conlflicting scheduling data.

    How test will be performed: Sample scheduling data and a sample reschedule request will be provided to the system. The test succeeds if a reschedule request is either accepted or denied and the schedule is updated or remains the same with no conlflicting scheduling data.

15. test-IG2

    Type: Non-Functional, Dynamic, Manual

    Initial State: The system is logged in to a commissioner level account.

Input: A user attempts to delete their account.

Output: A warning that you cannot delete the only commissioner level account.

How test will be performed: A commissioner level account will attempt to delete their account. The test succeeds if they get a warning that their account cannot be deleted since it is the only commissioner level account and the account is not deleted.

16. test-IG3

    Type: Non-Functional, Dynamic, Manual

    Initial State: The system is logged in to a captain level account.

    Input: A user attempts to contest a match score.

    Output: An option is visible to the user to contest a recorded match score.

    How test will be performed: A captain level account will attempt to contest a match score. The test succeeds if they are able to view an option to contest a recorded match score since they are a captain level account and they can view and contest a recorded match score.

17. test-IG4-1

    Type: Non-Functional, Dynamic, Manual

    Initial State: The system is logged in to a captain level account.

    Input/Condition: A user awaits a notification from an alert.

    Output: A notification is received by the user about an alert.

    How test will be performed: A captain level account will be sent a sample alert. The test succeeds if they receive a notification about the alert and they should observe if any errors or failures occur.

18. test-IG4-2

    Type: Non-Functional, Dynamic, Manual

    Initial State: The system is logged in to a captain level account.

Input/Condition: A user awaits a notification from a reschedule request.

Output: A notification is received by the user about a reschedule request.

How test will be performed: A captain level account will be sent a sample reschedule request. The test succeeds if they receive a notification about the reschedule request and they should observe if any errors or failures occur.

19. test-PV1

    Type: Non-Functional, Dynamic, Manual

    Initial State: The system is not logged in.

    Input/Condition: A user is not logged into an account.

    Output: No contact information is displayed to the user.

    How test will be performed: A user will not be logged into an account and will observe if any contact information can be seen or accessed. If no contact information is revealed to the user, the test succeeds.

20. test-PV2

    Type: Non-Functional, Dynamic, Manual

    Initial State: The system is not logged in.

    Input: A user creates an account.

    Output: A warning to keep your password secure, and a new account is made.

    How test will be performed: A new account will be created. If the user receives a warning to keep their password secure, the test succeeds.

4.2.7 Cultural Requirements

1. test-CL1

   Type: Non-Functional, Dynamic, Manual

Initial State: A user is creating an account for themselves on the solution.

Input/Condition: A user selects a gender option when creating their account.

Output/Result: A user is able to select a gender option that does not specify their gender when creating their account.

How test will be performed: A tester will be provided the solution and will be asked to create a new account. They will then be asked to select the gender option that allows them to not specify their gender and observe if their information is displayed correctly when they are finished creating their account or if any errors had occured.

4.2.8 Migration to the New Product

1. test-

4.2.9 User Documentation and Training

1. test-TR1

No test needed for this requirement.

4.3 Traceability Between Test Cases and Requirements

The name of our test cases are each stamped with the requirement number which they address. Each requirement in our SRS (1) has an associated test to ensure full coverage. If a requirement doesn't require a test it will be noted in section 4 above.

5 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]

5.1 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]

5.2 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]

5.2.1 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. ...

5.2.2 Module 2

...

5.3 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]

5.3.1 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:

5.3.2 Module ?

...

5.4 Traceability Between Test Cases and Modules

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

6 Appendix

6.1 Symbolic Parameters

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

6.2 Usability Survey Questions?

A Google Form will be provided to testers of the system to validate specific system tests outlined in the VnV plan. Questions in the Google Form will be the following:

1. Were you able to access the system?
    (a) Strongly Agree
    (b) Slightly Agree
    (c) Neutral
    (d) Slightly Disagree
    (e) Strongly Disagree
2. Was the system's user interface readable and easy to understand?
    (a) Strongly Agree
    (b) Slightly Agree
    (c) Neutral
    (d) Slightly Disagree
    (e) Strongly Disagree
3. Was the system's user interface easy to navigate?
    (a) Strongly Agree
    (b) Slightly Agree
    (c) Neutral
    (d) Slightly Disagree
    (e) Strongly Disagree

4. Was the system's user interface easy to use?
    (a) Strongly Agree
    (b) Slightly Agree
    (c) Neutral
    (d) Slightly Disagree
    (e) Strongly Disagree

5. If given inputs, were you able to receive the expected output from the system?
    (a) Strongly Agree
    (b) Slightly Agree
    (c) Neutral
    (d) Slightly Disagree
    (e) Strongly Disagree

6. Were there any problems/issues you ran into when interacting with the system? Write them down below.

7. Is there anything else about the system you would like us to know? Write your thoughts down below.

Appendix — Reflection

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?

2. What pain points did you experience during this deliverable, and how did you resolve them?

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.

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?

Team Reflection

1. 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.

Our team will need to collectively acquire a great deal of knowledge and skills to successfully complete the verification and validation of our project. In particular, knowledge on how to implement continuous integration using Github actions will be key to automating unit tests to run whenever changes are made to the code. Furthermore, the skills and knowledge needed to implement effective unit tests using pytest is the core of the verification and validation of the project. Additionally, understanding how to implement and properly analyze code coverage using tools like Coverage.py for Python and Jest for React will be key to ensuring our unit tests are robust and properly test all states of our code. Finally, learning how to properly setup and use linters like flake8 for Python and ESLint for React will be extremely helpful for catching bugs before we test, hopefully reducing the amount of time we spend debugging. Casra will tackle Github actions, Jung-woo will learn about pytest, Nicholas will look into code coverage tools, and Alex will handle linters.

2. 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?

   Regarding the knowledge area of CI and Github actions, 2 possible approaches to learning the skills required would be through official documentation from Github or online video tutorials. Casra will tackle learning about CI and Github actions from the official Github documentation because he believes that learning about a tool from the people that made it will include more thorough information than the learning material of a 3rd party (ie. a video tutorial). In terms of learning the knowledge and skills needed to create unit tests via pytest, 2 possible approaches to gaining the knowledge and skills required would be through the official pytest documentation or more general articles discussing the basics and best practices for unit testing. Jung-woo will

learn about unit testing through the more general articles because he believes that starting with the more basic knowledge and working his way up to more framework specific unit testing knowledge will allow him to have a better and deeper understanding of how to make proper unit tests. Regarding the knowledge area of code coverage and related tools like Coverage.py and Jest, 2 possible approaches to learning the skills required would be through online video tutorials, or testing out the tools yourself. Nicholas will learn about code coverage and the related tools by testing out the tools himself on sample code. This is because he believes that he will learn more quickly by trying out the tools himself rather than reading about how they work. Finally, in terms of learning the knowledge and skills needed to setup and work with linters like flake8 and ESLint, 2 possible approaches to gaining the knowledge and skills required would be through official tool documentation or referencing previous projects which used the tools. Alex will learn about linters and the related tools by referencing a previous project in which he used flake8. This is because he believes that by revisiting an old project he's worked on in the past it will help jog his memory on how he used linters previously.

Casra Ghazanfari – Reflection

1. What went well while writing this deliverable?

   Section 3.6: Automated Testing and Verification Tools went well while writing this deliverable because we had already discussed all these topics in the development plan's expected technology section. This allowed us to simply reference the development plan document and provide a high level overview of the information we previously discussed rather than spending time researching and writing in detail about the topic from scratch for the VnV document.

2. What pain points did you experience during this deliverable, and how did you resolve them?

   The main pain point we experienced during this deliverable was having to spend a large chunk of our time on a major revision on our requirements document. We did this revision because we felt our functional and access requirements were still incomplete as we worked through

some initial tests. Even though we had already done a major revision on our requirements document previously when working on our hazard analysis document, that revision was very NFR focused. Therefore we felt another revision was necessary to target our FRs and ARs. Ultimately, this process took a good chunk of our time working on the VnV document but was worth it to ensure the robustness of our requirements and their respective tests.

Nicholas Fabugais-Inaba – Reflection

1. What went well while writing this deliverable?

   When writing this deliverable, the things that went well were the organization of tasks amongst each team member. Particularly with the system tests, each team member was given a group of requirements to write system tests for regarding both the functional and non-functional requirements located in the SRS. This allowed us to accomplish our own individual work, while as a team, we were able to discuss the various plans outlined in the Plan section of the VnV document. With the SRS requirements having an adequate amount of details to work from, writing the system tests became very easy to trace back to the SRS requirements.

2. What pain points did you experience during this deliverable, and how did you resolve them?

   Some of the pain points from this deliverable were figuring out which tests were not covered by the SRS requirements. This was an issue as certain tests we realized we should include in our VnV plan were not addressed by the previously created SRS requirements. This meant the team had to go back and adjust or add new requirements to the SRS document, where we could then add system tests to appropriately address certain functionalities that would require sufficient testing by a user of the system.

Jung Woo Lee – Reflection

1. What went well while writing this deliverable?

   Writing the test cases helped me see how the product would shape to be, and allowed the team to make crucial changes to the requirements. They helped me verify if requirements were atomic and self-contained or not. It also helped understand the nature of some requirements as I considered the test types: manual vs. automatic and dynamic vs. static.

2. What pain points did you experience during this deliverable, and how did you resolve them?

   There trouble coming up with initial states of some tests in terms of what to include or exclude. I found that it was easy to overlook a crucial but obvious detail in this section of the tests. I also had trouble with thinking of inputs for the tests. One issue was trying to avoid as much implementation detail as possible. Another was when tests seemingly had no obvious input. Lastly, considering invalid cases caused some difficulty, as these are not explicitly mentioned by the requirements document, so this was the first time thinking about what should not happen instead of what should happen. These were all resolved either by more thorough thought, discussion with team members, or looking at examples to gauge how they could be written.

Alex Verity – Reflection

1. What went well while writing this deliverable?

   Writing section 4, especially the functional tests, helped fill in a lot of holes in the coverage of the SRS (1). I'm always glad when this happens as I believe it leads to a better final SRS document and I hope a smoother development workload as we have a better grasp on what our final solution will look like.

2. What pain points did you experience during this deliverable, and how did you resolve them?

Writing so many tests was definitely a pain point, as we needed to go back to our SRS and update our requirements. Particularly the functional requirements and access requirements. It is similar to what went well while writing this deliverable but finding more requirements is a double edged sword. We resolved it by filling out the SRS and fixing requirements that needed updating.