System Verification and Validation Plan for Software Engineering

Team 6, Pitch Perfect
Damien Cheung
Jad Haytaoglu
Derek Li
Temituoyo Ugborogho
Emma Wigglesworth

April 4, 2025

Revision History

Date	Version	Notes
Mar 7, 2025 1	2.0	Unit Tests
Oct 30, 2024 1	1.0	4.2 and Appendix

Contents

List of Tables

1 Symbols, Abbreviations, and Acronyms

symbol	description
RBAC	Role-Based Access Control. Used for defining user permissions
UI	User Interface
JWT	JSON Web Token. A compact way to transmit info securely
GSA	Graduate Students' Association. The association overseeing the league
SRS	Software Requirements Specification
MIS	Module Interface Specification
DD	Design Document
VnV	Verification and Validation
CI	Continuous Integration

This document defines the procedures, testing strategies, and methodologies that will be used to ensure that the platform meets its functional, performance, and user experience requirements. Verification and Validation are essential to establishing that the system is developed in accordance with its specifications and operates reliably within the designated environment. Roadmap of the VnV document:

- **General Information:** Provides a summary, objectives, challenges, and relevant documentation for the Verification and Validation plan.
- **Plan:** Outlines the team structure, testing plans for the requirements and design, automated testing tools, and the software validation plan.
- **System Tests:** Describes the system-level test cases for functional and nonfunctional requirements, including detailed tests for authentication, team management, scheduling, and additional aspects such as usability, performance, and security. It also includes a traceability matrix that maps test cases to requirements.
- **Unit Test Description:** Specifies the scope of unit testing, with detailed cases for both functional and nonfunctional requirements. Traceability between test cases and modules is also included here.

2 General Information

2.1 Summary

The software being tested is the McMaster GSA Softball League Management Platform. This platform is designed to support and streamline operations for the McMaster GSA softball league. Key features include team management, game scheduling, reporting functionalities, and communication tools. The platform aims to simplify tasks for league administrators, captains, and players by providing a centralized system for managing league information, tracking standings, scheduling games, and maintaining up-to-date team rosters.

2.2 Objectives

Primary

- **Build confidence in software correctness**: Ensure that core functionalities, including team management, game scheduling, and result reporting, operate as expected.
- **Demonstrate adequate usability**: Confirm that the system interface is intuitive and accessible for users with varied technical backgrounds, such as league commissioners, captains, and players.
- **Verify data integrity**: Ensure accurate and consistent data across all features, from player rosters to game results and standings.
- **Assess security**: Validate that user access levels and authentication are correctly implemented to secure sensitive data.

Out of Scope

- **Accessibility Testing**: Accessibility checks, such as screen reader compatibility, keyboard-only navigation, and compliance with accessibility standards (e.g., WCAG), will not be covered in this scope. This decision is made because the platform's primary user base does not require stringent accessibility support, and our focus is primarily on basic usability rather than comprehensive accessibility.
- **Performance and Load Testing**: Extensive load and stress testing to measure system performance under high user traffic will be omitted. Given the platform's limited expected user base and the controlled environment in which it will operate, performance considerations are minimal. Our team will assume that system responsiveness and speed are sufficient for a small to medium load without simulating peak traffic.
- **In-depth Security and Vulnerability Testing**: While security basics, such as user authentication and access control, will be verified, extensive security testing (e.g., penetration testing and vulnerability scanning) is outside our scope. This is due to resource constraints and the relatively low-risk nature of the application, which handles limited sensitive data and is intended for a small group of users within the league.

- **External Library and API Validation**: It is assumed that all libraries and APIs have been thoroughly tested by their developers, so they will not undergo additional validation in our testing scope. Our focus will be on integration rather than on validating the correctness of external components themselves.

2.3 Challenge Level and Extras

The challenge level we have established for this project is General. The extras we have described for our project include usability testing, user documentation, and design thinking. We believe these are strongly relevant to our project since it is a heavily user-centric web application that must incorporate many principles of human-computer interfacing. We will track our changes consistently with written documentation as well as version control (via Git and GitHub). We will incorporate design thinking when developing and designing our system and UI, and we will conduct comprehensive usability tests with existing/potential stakeholders to compile improvements and verify design decisions.

2.4 Relevant Documentation

- **Software Requirements Specification (SRS)** (**?**): The SRS defines the detailed functional and non-functional requirements of the system, providing a clear basis for testing each requirement. By mapping each test case back to the SRS, we can ensure complete coverage of all system specifications.

- **Hazard Analysis** (**?**): The Hazard Analysis identifies potential hazards associated with the software system, assessing their impact and likelihood. This document is critical for understanding risk factors that may affect system safety and reliability. By identifying and analyzing hazards early in the development process, we can implement strategies to mitigate risks and ensure that the system operates safely under expected conditions.

- **Development Plan Document** (**?**): The development plan contains architectural decisions and structural design rationale for the system, which are critical for validating design integrity and system stability.

This document guides us in identifying key design aspects that need testing to verify adherence to design principles and effective implementation of required functionalities.

- **Module Interface Specification (MIS)** (TBD): The MIS defines the interface details for each module, including input and output data formats, parameter types, and function signatures. This document ensures that modules correctly integrate with one another, supporting our VnV plan by allowing us to focus on the correctness of inter-module communication.

- **User Manual (UM)** (TBD): The UM outlines the user interface and operational aspects of the system. It serves as a reference for usability testing, as it details expected user workflows and assists in validating whether the system operates as anticipated from the end-user perspective.

3 Plan

3.1 Verification and Validation Team

V&V Team Lead: Derek

- Oversees the entire verification and validation process and ensures all activities are completed according to the plan.
- Coordinates with other members to align V&V activities with project deadlines.
- Reviews and approves test plans, reports, and results.
- Researches and ensures that the team has the tools needed for testing.

Developers: Emma, Damien, Temituoyo, Jad

- Develop code and conduct unit testing to ensure basic functionality
- Address any defects or issues they identified during testing
- Participate in code reviews and provide feedback to ensure code implementation is of high quality

- Assists in setting up automated testing frameworks

Supervisor: Dr. Jake Nease

- Ensures that requirement tests and requirements align with project needs

Individual Breakdown

1. **Verification Lead - Damien**
    - Responsible for overseeing all verification tasks, including:
        - Ensuring all test cases are executed as planned.
        - Validating that the test cases align with system requirements.
        - Maintaining the traceability matrix between requirements and test cases.
        - Reporting progress to the project manager.
2. **Validation Lead - Emma**
    - Responsible for ensuring the product meets user needs and client expectations:
        - Conducting usability testing with representative users.
        - Collaborating with stakeholders to confirm test results meet acceptance criteria.
        - Managing and documenting feedback from validation testing.
        - Ensuring functional and non-functional requirements are tested appropriately.
3. **Test Engineers - Derek**
    - Execute both verification and validation tasks under the guidance of the Verification and Validation Leads:
        - Write and execute test cases based on requirements and design specifications.
        - Report and document any defects found during testing.
        - Re-run test cases after defect fixes to confirm resolution (regression testing).

- Support validation testing by preparing test environments.

4. **Traceability Coordinator - Tuoyo**

   - Focused on maintaining the alignment between requirements and test cases:
     - Update the traceability matrix as new requirements or test cases are added.
     - Collaborate with the Test Engineers to ensure all requirements are covered by test cases.
     - Provide weekly updates on traceability coverage metrics.

5. **Defect Manager - Jad**

   - Manages the lifecycle of identified defects:
     - Triage defects based on priority and severity.
     - Track defect resolution and communicate updates to the team.
     - Ensure resolved defects are verified through regression testing.

3.2 SRS Verification Plan

Our approach for SRS verification involves understanding the following objectives:

- Validate that the SRS covers the project's functional and non-functional requirements
- Verify that requirements are clearly defined, consistent and testable
- Validate that requirements are feasible and achievable within project constraints

After understanding these objectives, we will prioritize the following steps:

1. Have a team meeting and brainstorm all use case scenarios and requirements before meeting with our supervisor. We can make a SRS checklist to do this.

2. Conduct a peer review with another team. For our capstone, another team is also doing the same project, so it will be best to work with them.
3. Book a meeting with our supervisor and present the following use cases and requirements our platform covers.
4. Discuss possible use cases and requirements with our supervisor. We can add these new ideas to our SRS checklist.

3.3 Design Verification Plan

Verification Methods:

- Code Reviews

  Team members can conduct code reviews regularly to focus on code readability, efficiency, and to ensure that our code meets design requirements and specifications.

- Automated Testing

  By using an automated testing framework, we can simulate real user interactions, such as signing up for the league. These testing scripts can be run quickly and save us time.

- Manual Testing

  We can conduct testing with users involved in the league to gather feedback and ensure that the platform meets user expectations.

- Performance Testing

  Check website performance under heavy load and traffic to meet user expectations. An example would be how the website runs when multiple users register teams at the same time.

- Security Testing

  Check for potential security issues, such as SQL injection to prevent data from being stolen.

- Data Verification

  Verify that data is being stored, retrieved, and updated accurately in our database to prevent errors and meet user expectations.

- Documentation Review

  Update documentation frequently, such as user guides, system diagrams, etc. to enforce updated knowledge throughout the team.

3.4 Verification and Validation Plan

Checklist:

- Code Reviews
  - Does our code meet readability and efficiency standards?
  - Are there code sections we can optimize?
  - Does the implementation match the design specifications?
- Automated Testing
  - Does our automated script cover the basic functionalities of our design specifications?
  - Are there edge cases missing?
- Manual Testing
  - Do the manual tests reflect real scenarios?
  - Is user feedback collected effectively and analyzed?
- Performance Testing
  - Does our platform handle high traffic without difficulties?
  - Do we have performance benchmarks and are they met?
- Security Testing
  - Did we test against common vulnerabilities?

  - Is sensitive data properly handled and protected?

- Data Verification
  - Is data transactions being done correctly?
  - Does our system maintain data accurately?

- Documentation Review
  - Are our documents up to date?
  - Are user guides easy to understand?

Milestone	**Description**	**Completion**
Code Reviews	Regular reviews focusing on readability and efficiency	Week 2
Automated Testing	Implement and run automated test scripts	Week 4
Manual Testing	Conduct user testing and gather feedback	Week 6
Performance Testing	Test website under heavy load	Week 8
Security Testing	Perform security checks and vulnerability assessments	Week 10
Data Verification	Ensure data is accurate and properly handled	Week 12
Documentation Review	Update user guides and technical documentation	Week 14

Table 1: Milestone Schedule

3.5 Implementation Verification Plan

- Code Walkthroughs

  Developers of a segment of code will share and present it to peers. The main focus here is design principles and coding standards.

- Code Inspections

  Peers will inspect code to find possible ways for optimization.

- Static Analyzers

  Use linting tools, such as ESLint to detect syntax errors and uphold standard code practices.

3.6 Automated Testing and Verification Tools

Primary Testing Tools:

- **Playwright**: End-to-end testing framework for cross-browser testing
- **Jest**: Unit testing framework for both frontend and backend
- **React Testing Library**: Component testing for React frontend

Code Quality Tools:

- **ESLint**: For JavaScript/TypeScript linting and code style enforcement
- **Prettier**: For consistent code formatting

Continuous Integration:

- **GitHub Actions**: For automated testing, documentation generation, and deployment
- **Jest Coverage Reports**: For tracking test coverage metrics

Performance Testing:

- **Chrome DevTools**: For frontend performance profiling
- **Node.js Profiler**: For backend performance analysis

Security Testing:

- **OWASP ZAP**: For security vulnerability scanning
- **Helmet.js**: For HTTP security headers

3.7 Software Validation Plan

Methods:

- Requirement-Based Testing

  Create test cases based on requirements to verify that each functionality works as expected.

- Regression Testing

  Ensure basic functionality remains unaffected when developing new features by running a script that tests all the basic functionalities before updates are pushed.

- Device Testing

  Validate that the platform works across various devices, such as smartphones, tablets, desktops, etc.

- Cross-browser Testing

  Validate that the platform works across various browsers, such as Chrome, Firefox, Safari, etc.

4 System Tests

This section goes over the tests for both functional and non-functional requirements of the platform. Functional requirements tests cover core features like authentication, team management, and scheduling. These tests are focused on making sure the system does exactly what it's supposed to, based on the requirements laid out in the SRS.

The non-functional requirements tests are split into categories to check additional qualities of the system: Look and Feel, Usability and Humanity, Performance, Operational and Environmental, Maintainability and Support, and Security. These tests make sure the platform isn't just functional but also user-friendly, reliable, compatible across devices, and secure. Each section breaks down the tests and criteria we'll use to evaluate these areas.

4.1 Tests for Functional Requirements

This section is divided into different areas to cover subsets of the functional requirements comprehensively. The following tests cover core features like authentication, team management, and scheduling. References to the SRS are provided to show the alignment with documented requirements.

It should be noted that when `Roles` are described doing actions in the functional tests (e.g. "Captain inputs..."), these are assumed a tester acting in place of the role.

4.1.1 Authentication and Access Control

This section includes tests related to the authentication of users and the enforcement of role-based access control. These tests cover requirements related to login functionality and role-specific permissions as outlined in the SRS data model. This section coveres SRS Req #1

Login with Invalid Credentials

1. FR-1
   Control: Manual

   Initial State: System at login screen, ready to accept credentials.

   Input: Invalid `UserID` and a combination of valid `UserID` + Invalid `Password` for commissioner, captain, and player accounts.

   Output: Unsuccessful login. The system is ready to accept new credentials.

   Test Case Derivation: Invalid login credentials should result in an unsuccessful login and the user should be able to try again.

   How Test Will Be Performed: Tester will attempt to log in with an invalid `UserID` and `Password`. Verify that all invalid logins are unsuccessful.

   Requirements Covered: SRS Req #1 (authentication).

Login with Valid Credentials

1. FR-2
   Control: Manual

   Initial State: System at login screen, ready to accept credentials.

   Input: `UserID` and `Password` for valid commissioner, captain, and player accounts.

   Output: Successful login for valid credentials.

Test Case Derivation: Valid login credentials should result in a successful login to the system. This is also how the system will differentiate users to perform actions requiring Role permissions.

How Test Will Be Performed: Tester will log in with a valid `UserID` and Password for each `Role` (Player, Captain, Commissioner) and verify access to the platform after successful login under the correct account.

Requirements Covered: SRS Req #1 (authentication).

4.1.2 Team Management

This section includes tests for the functionality of team creation and management. These tests verify that captains can create teams with unique names, manage join requests, and prevent duplication errors. This section also includes tests for the functionality allowing players to browse and request to join teams. These tests confirm that players can request to join a team and that requests are accurately reflected to captains. This section covers SRS Req #5 and #6.

Captain Registering a Team

1. FR-3
   Control: Manual

   Initial State: User is logged in with Role set to Captain, with no existing team associated with `CaptainID`. System is on the create team page, with at least one team registered in the database.

   Input: Enter a `TeamName` that already exists in the database and a Division.

   Output: Unsuccessful creation of a team with a duplicate `TeamName`. Error message displayed to Captain addressing that the `TeamName` already exists.

   Test Case Derivation: As a general rule, there are to be no duplicate `TeamName`. Captains should not be able to create a team with a `TeamName` that is already taken

   How Test Will Be Performed: Tester (with Captain account) submits a request to create a new team using a duplicate `TeamName`. Verify that the team is not added to the database.

Requirements Covered: SRS Req #5 (team creation).

2. FR-4
   Control: Manual

   Initial State: User is logged in with Role set to Captain, with no existing team associated with `CaptainID`. System is on the create team page.

   Input: Enter a unique TeamName and Division.

   Output: Successful team creation after submission.

   Test Case Derivation: With valid input, Captains should be able to register their team into the system.

   How Test Will Be Performed: Captain creates a valid new team. Verify that the team is created and added to the database. New entry should include `TeamID, TeamName, Division, CaptainID, Roster`.

   Requirements Covered: SRS Req #5 (team creation).

3. FR-5
   Control: Manual

   Initial State: User logged in with Role set to Captain, with an existing team associated with `CaptainID`. System is on the create team page.

   Input: Enter a new TeamName and Division.

   Output: Unsuccessful creation of a team. Error message displayed for multiple teams under one `CaptainID`.

   Test Case Derivation: As a general rule, Captains only have one team. Captains should not be able to create a new team when they already have one registered under their `CaptainID`.

   How Test Will Be Performed: Captain attempts to create a second team. Verify that the team is not entered in the database.

   Requirements Covered: SRS Req #5 (team creation).

Joining a Team

1. FR-6
   Control: Manual

Initial State: User logged in with Role set to Player, without an existing team associated with `PlayerID`. Another user logged in as Captain of `TeamName`.

Input: Player requests to join `TeamName`.

Output: Player's request is sent to the team's Captain.

Test Case Derivation: Players should be able to send join requests to Captains for review and approval.

How Test Will Be Performed: Player requests to join a team. Verify that the associated Captain recieves the request (Refer to **FR-7**).

Requirements Covered: SRS Req #6 (joining teams).

2. FR-7
   Control: Manual

   Initial State: User logged in with Role set to Captain, with an existing team associated with `CaptainID`. A player has requested to join the associated team.

   Input: Captain receives and approves request.

   Output: `PlayerID` is added to `Roster` of the team associated with `TeamName`.

   Test Case Derivation: Players should be added to a team after a request is approved.

   How Test Will Be Performed: Captain approves a pending player's join request. Verify the team updates accordingly: `Roster` updated in database and the player should have permissions to view team-specific information.

   Requirements Covered: SRS Req #6 (joining teams).

4.1.3 Game Scheduling and Reporting

This section tests game scheduling and rescheduling requests, focusing on the interaction between captains and commissioners for adjusting game schedules as defined in SRS Req #7, #8, #11, and #12.

Automated Season Schedule

1. FR-7
   Control: Manual

   Initial State: Teams have been registered in the system, each with `TeamIDs` and preferences for game days/times. No current season schedule exists.

   Input: Set up a sample database file containing:

   - Team Availability: Preferences for game days and times.
   - Division Assignments: Each `TeamID` is assigned a division.
   - Game Count Requirement: Ensures each team has an equal number of games.
   - Available Slots: The availability of the event space.

   Output: The system generates an appropriate season game schedule.

   Test Case Derivation: The system needs to be able to generate an initial schedule for the season based on team availability and preferences.

   How Test Will Be Performed: Load the database with team availability and divisional information. Run the schedule generation process. Refer to **TPERF-2** for performance test of the generated schedule.

   Requirements Covered: SRS Req #7 (Schedule Automation).

Captain Game Reporting

1. FR-8
   Control: Manual

   Initial State: Season schedule is generated and displayed, with captain logged in. System is on game report page.

   Input: Captain submits game results (`Results, ScoreTeamA, ScoreTeamB`) or forfeits for a selected game.

   Output: Game report is updated and Standings are updated (`TeamID, Wins, Losses, Ties, Rank`).

   Test Case Derivation: Captains should be able to report game results which should be reflected in the platform's standings/reporting.

How Test Will Be Performed: Captain reports games. Verify that the report is reflected in database and that standings are automatically updated accordingly.

Requirements Covered: SRS Req #12 (Reporting).

Rescheduling

1. FR-9
   Control: Manual

   Initial State: Season schedule is generated and displayed, with commissioner and captain logged in. System is on scheduling page.

   Input: Captain submits a reschedule request in an open slot. The request is valid and associated with the appropriate `ScheduleID, GameID, and SlotNumber`.

   Output: Request is sent to Commissioner.

   Test Case Derivation: Rescheduling requests from captains should be sent to the Comissioner for review and approval.

   How Test Will Be Performed: Captain initiates reschedule request. Verify that the request is sent to the commissioner for approval.

   Requirements Covered: SRS Req #8 (rescheduling).

2. FR-10
   Control: Manual

   Initial State: Commissioner logged in. System is on schedule page with a scheduled game entry available.

   Input: Commissioner applies an override (cancel or reschedule) on an existing `GameID` or approves a reschedule request.

   Output: Override is reflected in the schedule. Notifications are sent to all IDs in `Roster` of teams associated with the overridden game.

   Test Case Derivation: Schedule changes should be reflected in all affected user's view and they should be notified of the changes.

   How Test Will Be Performed: Commissioner applies an override on an existing game. Verify the game change (`Schedule view, SlotNumber`,

and `Available Slots` are updated). Verify notifications are received by both teams.

Requirements Covered: SRS Req #11 (game overrides).

4.1.4 Announcements

This section includes tests for league-wide announcements and commissioner game overrides, ensuring that all users receive notifications of changes and announcements, per SRS Req #9 and #10.

Posting Announcements

1. FR-11
   Control: Manual

   Initial State: Commissioner logged in. System is on announcement page.

   Input: Commissioner posts a league-wide announcement to be visible across platform views.

   Output: Announcement is visible to all users.

   Test Case Derivation: Commissioners' posts should be viewable by all users.

   How Test Will Be Performed: Commissioner posts announcement; verify visibility for all `Roles`.

   Requirements Covered: SRS Req #9, #10 (announcements).

...

4.1.5 Area of Testing2

...

4.2 Tests for Nonfunctional Requirements

4.2.1 Look and Feel

Consistent and Usable UI

1. TLF-1

   Control: Manual

   Initial State: Web application is launched.

   Input: Tester navigates through primary views (including login, team management, and schedule viewing)

   Output: The platform's interface should be modern, intuitive, and visually consistent across all views.

   Test Case Derivation: Ensures a consistent, modern interface as specified.

   How test will be performed: Manually inspect visual consistency across pages on different devices. This can be included in the results of the usability survey as well as remaining consistent with the stylings and design library used.

   Requirements Covered: SRS Req 10.1.

4.2.2 Usability and Humanity

Localization and Easy Navigation

1. TUH-1

   Control: Dynamic, Manual

   Initial State: Web application is launched, user is on login page.

   Input: Tester is asked to navigate through different platform views (login, announcements, standings, schedule)

   Output: Tester successfully navigates to each view within MIN_NAVTIME minutes with no prior experience or external help.

   Test Case Derivation: Ease of use is achieved if new users can complete essential navigation quickly and independently, as specified.

   How test will be performed: Conduct a observational study of usability with a MIN_TESTERS participants who have no prior experience with the platform. Record the time taken for each view. Following the tasks,

participants will complete a survey (in Appendix) to provide feedback on task difficulty and ease of navigation.

Requirements Covered: SRS Req 11.1.

2. TUH-2

   Control: Manual

   Initial State: Web application is launched, user is on login page.

   Input: Tester views date and time fields, metric system measurements, and content language.

   Output: Tester verifies platform displays date and time in Canadian format, uses the metric system, and all text is in Canadian English.

   Test Case Derivation: Adhering to Canadian localization standards ensures that users are comfortable with the displayed information format.

   How test will be performed: Verify date, time, and measurement units across the platform views. Ask usability testers to confirm that information formats are clear and match Canadian standards. Feedback survey included in Appendix.

   Requirements Covered: SRS Req 11.2.

3. TUH-3

   Control: Dynamic, Manual

   Initial State: Web application is launched, user is on login page.

   Input: Tester is asked to perform certain tasks within the platform (login, create team, request reschedule)

   Output: Tester completes each task within an average of AVG_TASK_TIME minutes with no prior experience, while easily understanding and utilizing in-app help (navigation instructions, tooltips, and help documentation) to complete tasks.

   Test Case Derivation: Effective learning support is demonstrated if new users can understand and perform basic tasks using the platform's in-app guidance.

How test will be performed: Conduct a observational study of usability with MIN_TESTERS participants who have no prior experience with the platform. Record the time taken for each task completion. Following the tasks, participants will complete a survey (in Appendix) to provide feedback on task difficulty and provided guidance.

Requirements Covered: SRS Req 11.3-11.4

4.2.3 Performance

Accurate Data Representations

1. TPERF-1

   Type: Automated

   Initial State: Platform is set up with sample data for standings and scheduling.

   Input/Condition: Tester views league standings.

   Output/Result: The standings are calculated (as mentioned in FR) with 100% accuracy.

   How test will be performed: Create test cases with predefined standings. Use an automated script to perform calculations and check results against expected outcomes, ensuring all calculations are accurate.

   Requirements Covered: SRS Req 12.1, 12.3.

2. TPERF-2

   Type: Automated, Manual

   Initial State: Platform is set up with sample data (teams and preferences) for scheduling.

   Input/Condition: Tester views league schedule.

   Output/Result:

   - The displayed schedule shows that all teams have a balanced number of scheduled games, ensuring that no team has a game count that differs by more than MAX_GAME_DIFF games from any other team.

- The displayed schedule shows that all teams play 100% of their games within their division.
- The displayed schedule is conflict-free, considering time and location.
- The displayed schedule is optimized based on matching team preferences.

How test will be performed: The platform is prepared with a sample dataset containing multiple teams and generates a schedule, ensuring each team is assigned games within the schedule. An automated script is used to capture the output schedule and iterate through each team's scheduled games to verify the correctness of divisional match-ups, the balance of games, and log any conflicts.

Requirements Covered: SRS Req 12.4.

3. TPERF-3

   Type: Dynamic, Automatic

   Initial State: The web application is launched.

   Input/Condition: A pair of new login credentials are added.

   Output/Result: Format of credentials in database.

   How test will be performed: An automated script is used to add a pair of new credentials and check whether the information is stored in the database.

   Requirements Covered: SRS Req 12.2.

4. TPERF-4

   Type: Manual

   Initial State: The platform is operational with a form ready for submission (e.g., team registration, game scheduling).

   Input/Condition: Tester fills out the form with invalid data (e.g., missing required fields, incorrect data formats).

   Output/Result:

The platform highlights the fields with errors (e.g., required fields that are empty, invalid email formats) and displays appropriate error messages next to each highlighted field. The form submission is prevented until all errors are corrected.

How test will be performed:

Testers will intentionally leave required fields blank, enter invalid data formats and submit the form. Testers will verify that the fields with errors are highlighted visually and informative error messages appear next to the problematic fields, indicating the nature of the errors. Testers will confirm that the information in the form is not submitted into the database until the errors are corrected and the form is resubmitted, showing a success message upon submission

Requirements Covered: SRS Req 12.4.

4.2.4 Operational and Environmental

Accessability

1. TOPE-1

   Control: Manual

   Initial State: Platform launched on desktop, tablet, and smartphone.

   Input: Test and access all features and navigate on each device type.

   Output: Platform should be fully accessible on all device types.

   Test Case Derivation: Ensures multi-device compatibility.

   How test will be performed: Manual testing of the platform features, responsiveness, and navigations on desktop, tablet, and smartphone.

   Requirements Covered: SRS Req 13.1.

2. TOPE-2

   Control: Manual

   Initial State: Platform open in Chrome, Firefox, Safari, and Edge.

   Input: Access all features in each browser.

Output: Platform displays correctly across major browsers.

Test Case Derivation: Confirms browser compatibility.

How test will be performed: Testers review platform views and features in each browser.

Requirements Covered: SRS Req 13.2-13.3.

4.2.5 Maintainability and Support

1. TMS-1

   Type: Manual

   Initial State: The platform is launched.

   Input/Condition: Tester is asked to contact support.

   Output/Result: Support email receives help request from tester without any external help.

   How test will be performed: MIN_TESTERS Tester are asked to reach support through email and record any difficulties encountered.

   Requirements Covered: SRS Req 14.1, 14.2.

4.2.6 Security

Data Access Governance

1. TSEC-1

   Type: Manual

   Initial State: Platform is launched, and user roles are assigned.

   Input/Condition: Tester logs in as an Administrator, Team Manager, and Player and verifies correct access.

   Output/Result:

   - The Player role has restricted access with basic features (ex. viewing schedule, joining teams)

- The Captain role has access to additional team management features.
- The Commissioner has access to all administrative features.

How test will be performed: Manually log in with a dummy user role for each level and attempt to access various features. Verify that the access levels match the defined role-based access control requirements.

Requirements Covered: SRS Req 15.1, 15.2.

2. TSEC-2

   Type: Manual

   Initial State: Platform is ready for data entry.

   Input/Condition: Tester attempts to submit invalid information into the database (ex. team roster with missing player names, invalid team or game information).

   Output/Result: The system displays helpful error messages and prevents data entries - no data should be submitted or saved for invalid inputs, preventing any unwanted database behaviour or liability.

   How test will be performed: Enter invalid data into the respective forms and submit. Observe the system's responses and ensure appropriate validation messages are displayed.

   Requirements Covered: SRS Req 15.1, 15.2.

3. TSEC-3

   Type: Manual

   Initial State: Platform login page is loaded, and the database connection is active.

   Input/Condition: Tester inputs SQL injection strings into form fields such as the username and password fields.

   Output/Result:

   - The platform rejects input containing SQL injection patterns.

- No unintended database behavior occurs (e.g., unauthorized login or data loss).
- Error messages are displayed without revealing system vulnerabilities (e.g., "Invalid credentials" instead of SQL syntax errors).

How test will be performed:

(a) Navigate to the login page and enter SQL injection strings in the username and password fields.

(b) Attempt to log in and observe the platform's behavior.

(c) Verify that no unauthorized access or system behavior is triggered.

(d) Review server logs to ensure the application logs the attempt without exposing database details.

4.2.7 Cultural

1. TCU-1

   Type: Manual

   Initial State: Platform is launched in default settings.

   Input/Condition: Tester views platform content, date, and time information.

   Output/Result: All text is displayed in Canadian English, with time shown in the appropriate Hamilton, Ontario, Canada time zone (EST).

   How test will be performed: Manually inspect the platform to ensure text and date/time formats match Canadian English and the local time zone.

   Requirements Covered: SRS Req 16.1.

4.2.8 Compliance

Account Deletion and Usage Standards

1. TC-1
   Type: Functional, Manual

Initial State: Dummy user account exists, with personal data such as email and phone number stored in the platform.

Input/Condition: User initiates a request to delete their account and associated data.

Output/Result: All personal data associated with the account is removed from database and platform, and user receives confirmation of data deletion.

How test will be performed: Manually delete an account and verify that all associated data is removed. Log any unexpected time elapsed to ensure security and compliance in appropriate time.

Requirements Covered: SRS Req 17.1.

2. TC-2

Type: Manual

Initial State: Platform is launched.

Input/Condition: Tester visually inspect the platform against W3C web standards to verify adherence to industry standards (readable fonts, accessible colors, and clear navigation).

Output/Result: Tester verifies platform's adherence to W3C web standards.

How test will be performed: Manually inspect the interface against W3C web standards to verify font readability, contrast, and navigation.

Requirements Covered: SRS Req 17.2.

...

4.3 Traceability Between Test Cases and Requirements

Note : Requirement IDs are from SRS For more, refer to requirements in SRS Document ()

Requirement ID	Test Case ID(s)	Description of Co
FR-1	FR-1: Login with Invalid Credentials, FR-2: Login with Valid Credentials	Tests login functio with valid and invalid c
FR-2	FR-3: Captain Registering a Team, FR-4: Captain Team Duplication, FR-5: Captain Multiple Teams	Ensures unique team and duplication prev
FR-3	FR-6: Player Joining a Team, FR-7: Captain Approving Join Request	Covers team joining rec approvals.
FR-7	FR-7: Automated Season Schedule, TPERF-2: Schedule Performance Validation	Verifies automated s generation and perfo
FR-8	FR-8: Captain Game Reporting, Game Results Submission	Ensures captains can game results and handle r
FR-9	FR-9: Rescheduling Request, Schedule Update Notification	Verifies rescheduling r
FR-10	FR-10: Admin Schedule Override, FR-11: Announcement Posting	Validates admin abi override schedules and post a
NFR-11.1	TUH-1: Localization and Easy Navigation, TUH-2: Task Completion Time, TUH-3: Navigation Assistance	Tests user ease of nav through the platfo
NFR-12.4	TPERF-1: Accurate Data Representations, TPERF-2: Schedule Performance Validation	Verifies schedule accura performance constr
NFR-13.2	TOPE-1: Cross-browser Compatibility, TOPE-2: Web Standards Compliance	Confirms adherence standards and cross-browser
NFR-15.1	TSEC-1: Data Access Governance, TSEC-2: Input Validation, TSEC-3: SQL Injection Prevention	Validates security mea access control
NFR-16.1	TCU-1: Canadian Localization	Ensures proper localiza cultural compliar
NFR-17.1	TC-1: Account Deletion Compliance, TC-2: Web Standards Adherence	Verifies compliance with and data manager

Table 2: Traceability Between Test Cases and Requirements

5 Unit Test Description

5.1 Unit Testing Scope

The primary focus of unit testing in this project is on verifying the correctness of the basic backend models. These unit tests ensure that the fundamental data structures and validation rules within our system function correctly before integrating them into higher-level functionalities. Since more complex interactions, (scheduling, game assignments, authentication flows) will be tested through system and integration tests, unit tests will focus on the core behavior of the database models.

Thus, our unit test structure will focus on the main classes used in our application and we allow Functional and Non-Functional rquirements to be tested in other methods (Manual, System, Integration)

- **Backend Testing**:
  - Jest as the primary testing framework
  - MongoDB Memory Server for database testing
  - Supertest for API endpoint testing
  - Coverage reporting through Jest's built-in coverage tools
- **Frontend Testing**:
  - Jest with React Testing Library
  - Jest DOM for DOM-specific assertions
  - Component testing for React components
  - Integration with GitHub Actions for CI/CD

5.2 Code Coverage Goals

- Minimum 80% coverage for all backend models
- Minimum 70% coverage for backend controllers
- Minimum 60% coverage for frontend components
- Coverage reports generated on every test run

5.3 Tests for Models/Classes

5.3.1 Player Model

Testing the Player model ensures that the basic structure and validation rules for player accounts are functioning correctly and allows confidence in more complex features involving players, including team creation, team joining, etc.

1. test-player-creation

   Type: Functional, Dynamic, Automatic

   Initial State: Database is empty or contains other player records.

   Input: A valid player object with required fields (e.g., name, email, team ID).

   Output: The player is successfully created in the database with a unique ID.

   Test Case Derivation: Ensures the basic creation functionality of the Player model works and the database correctly stores new players.

   How test will be performed: Insert a player object into the database and verify that the player exists by querying the database.

2. test-player-uniqueness

   Type: Functional, Dynamic, Automatic

   Initial State: A player with the same email already exists in the database.

   Input: A new player object with the same email.

   Output: The database rejects the creation due to a uniqueness constraint violation.

   Test Case Derivation: Ensures that email uniqueness is enforced at the database level.

   How test will be performed: Attempt to insert a duplicate player and expect an error response.

5.3.2 Team Model

1. test-team-creation

   Type: Functional, Dynamic, Automatic

   Initial State: Database is empty or contains other teams.

   Input: A valid team object with required fields (e.g., name, division, captain ID).

   Output: The team is successfully created in the database with a unique ID.

   Test Case Derivation: Ensures teams can be properly created and stored.

   How test will be performed: Insert a team object into the database and verify its existence.

2. test-team-uniqueness

   Type: Functional, Dynamic, Automatic

   Initial State: A team with the same name already exists in the database.

   Input: A new team object with the same name.

   Output: The database rejects the creation due to a uniqueness constraint violation.

   Test Case Derivation: Ensures that team names are unique within the system.

   How test will be performed: Attempt to insert a duplicate team and expect an error response.

...

5.3.3 Game Model

1. test-game-creation

   Type: Functional, Dynamic, Automatic

Initial State: Database is empty or contains other games.

Input: A valid game object with required fields (e.g., date, time, field, teams).

Output: The game is successfully created in the database.

Test Case Derivation: Ensures games can be properly created and stored.

How test will be performed: Insert a game object and verify its existence in the database.

5.3.4 Gameslot Model

1. test-gameslot-uniqueness

   Type: Functional, Dynamic, Automatic

   Initial State: A game slot with the same date, time, and field already exists.

   Input: A new game slot with identical attributes.

   Output: The database rejects the creation due to a uniqueness constraint violation.

   Test Case Derivation: Ensures that game slots cannot have duplicate entries.

   How test will be performed: Attempt to insert a duplicate game slot and expect an error.

5.3.5 Schedule Model

1. test-schedule-creation

   Type: Functional, Dynamic, Automatic

   Initial State: Database is empty or contains other schedule entries.

   Input: A valid schedule object with required fields (e.g., season ID, team ID, game slots).

   Output: The schedule is successfully created and stored.

Test Case Derivation: Ensures schedules can be generated and stored properly.

How test will be performed: Insert a schedule object and query the database for validation.

...

5.4 Traceability Between Test Cases and Modules

Test Case Mapping to Modules

The following table outlines the relationship between the test cases and the respective modules:

Coverage of Anticipated Changes

The following anticipated changes (AC) are covered by corresponding test cases:

- **AC1:** Enhanced Authentication Features → TC-02 (Authentication Module Testing)
- **AC2:** Session Management Updates → TC-02 (Authentication Module, Token Handling)
- **AC3:** Database Schema Adjustments → TC-11 (Backend Module, Data Integrity)
- **AC4:** User Interface (UI) Enhancements → TC-01 (UI Testing for Compatibility and Responsiveness)
- **AC5:** Integration of New Scheduling Algorithms → TC-12 (Scheduling Algorithm, Game Assignments)
- **AC6:** Data Security and Privacy Updates → TC-11 (Backend Module, API Security Testing)
- **AC7:** Improved Notification System → TC-10 (Notification Delivery and Response)

Test Case ID	Module Under Test	Description of Test Case
TC-01	User Interface (M1)	Tests login functionality with valid and invalid credentials.
TC-02	Authentication (M2)	Validate login, logout, and role-based access control (RBAC) functionalities.
TC-03	Team Management (M3)	Ensure proper team creation, joining, and waiver processes.
TC-04	Game Management (M4)	Validate game creation, score updating, and game status tracking.
TC-05	Announcements (M5)	Check announcement creation, delivery, and retrieval for users.
TC-06	Standings (M6)	Validate standings calculations and rankings update mechanisms.
TC-07	Scheduling (M7)	Ensure game scheduling adheres to team preferences and constraints.
TC-08	Waiver Module (M8)	Verify waiver access, completion, and validation processes.
TC-09	PlayerT Module (M9)	Test individual player data handling and integration with teams.
TC-10	Notification Module (M12)	Ensure notifications are sent and received correctly by users.
TC-11	Backend Module (M13)	Verify API calls, data storage integrity, and security compliance.
TC-12	Scheduling Algorithm (M14)	Validate scheduling logic to ensure optimal game assignments.
TC-13	Reschedule Request Module (M19)	Test the rescheduling feature, ensuring proper slot selection and game updates.

Table 3: Traceability between Test Cases and Modules

5.5 Test Data

The test data will be managed through the following approaches:

- **Development Data:** Using MongoDB with sample data for development and testing
- **Data Validation:** Ensuring data integrity through MongoDB schema validation
- **Test Environment:** Separate test database to prevent interference with development data

5.6 Test Environment Setup

The test environment will be configured as follows:

- **Development Environment:**
  - Node.js with Express backend
  - MongoDB database
  - React frontend with TypeScript
- **Testing Tools:**
  - Jest for unit testing
  - Playwright for end-to-end testing
  - ESLint for code quality

5.7 Test Execution

The test execution process will be managed as follows:

- **Automated Testing:**
  - Unit tests run on every pull request
  - End-to-end tests for critical user flows
  - Code quality checks through ESLint
- **Manual Testing:**

  - Usability testing with 5 participants
  - Cross-browser testing
  - Mobile responsiveness testing

- **Test Results:**
  - Test coverage reports from Jest
  - Usability testing feedback and observations
  - Bug tracking through GitHub Issues

6 Unit Testing Summary

6.1 Backend Unit Testing

Our backend unit testing is conducted using Jest and executed via npm test in the backend directory. The test files are located in the backend/test folder. These unit tests focus on verifying the fundamental functionality of our backend models, ensuring they behave as expected when interacting with MongoDB.

6.1.1 Player Model Tests

- **test-player-creation**: Verifies that the player model can be created with valid input and default fields are set correctly.
- **test-player-uniqueness**: Verifies that the player model enforces uniqueness for the email field.
- **test-player-validation**: Ensures proper validation of required fields and data types.

6.1.2 Team Model Tests

- **test-team-creation**: Verifies that the team model can be created with valid input and default fields are set correctly.
- **test-team-uniqueness**: Ensures team names are unique within a season.
- **test-team-validation**: Validates required fields and data types.

6.1.3 Game Model Tests

- **test-game-creation**: Verifies that a game can be created with valid input and default fields are set correctly.
- **test-game-validation**: Ensures proper validation of game-related fields.
- **test-game-updates**: Verifies that game updates (scores, status) are handled correctly.

6.1.4 Schedule Model Tests

- **test-schedule-creation**: Verifies that a schedule can be created with valid input.
- **test-schedule-validation**: Ensures proper validation of schedule-related fields.
- **test-schedule-updates**: Verifies that schedule updates are handled correctly.

6.2 Frontend Unit Testing

Our frontend unit testing is conducted using Jest and React Testing Library. The test files are located in the frontend/src/test directory.

6.2.1 Component Tests

- **test-login-component**: Verifies that the login form renders correctly and handles user input.
- **test-team-management**: Ensures team management components function as expected.
- **test-schedule-view**: Verifies that the schedule view renders and updates correctly.

6.2.2 Integration Tests

- **test-authentication-flow**: Verifies the complete authentication process.
- **test-team-creation-flow**: Ensures the team creation process works end-to-end.
- **test-game-scheduling-flow**: Verifies the game scheduling process.

7 Feedback and Changes Documentation

7.1 Supervisor Feedback Implementation

- **Team Invitation System**: Based on supervisor feedback, we modified the team joining process to only allow captains to invite players, rather than allowing players to request to join teams. This change was implemented to prevent spam and improve the user experience for captains.
- **Game Result Submission**: Simplified the game result submission process by removing the need for captains to manually input game results (Win/Lose/Tie). Instead, results are automatically calculated based on scores.
- **User Interface Consistency**: Addressed feedback regarding inconsistent color usage (yellow appearing randomly) by implementing a standardized color scheme across all views.

7.2 User Testing Feedback Implementation

- **Navigation Flow**: Based on feedback from 5 test users, we improved the navigation flow to ensure all main views are accessible within `MAX_NAVTIME` seconds.
- **Mobile Responsiveness**: Enhanced mobile responsiveness based on user feedback, ensuring consistent experience across devices.
- **Form Validation**: Implemented stricter validation for email addresses and other form fields based on user testing feedback.

7.3 Team Feedback Implementation

- **Testing Strategy**: Modified our testing approach based on team feedback, focusing more on manual testing for certain features where it proved more effective than automated testing.
- **Documentation Updates**: Enhanced documentation based on team feedback to ensure clarity and completeness.
- **Code Organization**: Restructured code organization based on team feedback to improve maintainability.

7.4 TA/Instructor Feedback Implementation

- **Test Coverage**: Increased test coverage based on feedback, particularly for edge cases and error handling.
- **Documentation Standards**: Improved documentation to meet the required standards for clarity and completeness.
- **Code Quality**: Enhanced code quality through implementation of suggested improvements in error handling and validation.

7.5 Traceability of Changes

Table 4: Traceability of Changes to Feedback Sources

Change	**Feedback Source**	**Reason for Change**
Modified team joining process	Supervisor	Prevent spam and improve captain experience
Simplified game result submission	Team	Improve user experience and reduce errors
Standardized color scheme	User Testing	Address visual consistency issues
Enhanced mobile responsiveness	User Testing	Improve cross-device experience

Continued on next page

Table – Continued

Change	Feedback Source	Reason for Change
Improved form validation	User Testing	Prevent invalid data entry
Modified testing strategy	Team	Optimize testing efficiency
Increased test coverage	TA/Instructor	Ensure comprehensive testing
Enhanced documentation	TA/Instructor	Meet documentation standards

8 Test Maintenance

The test suite will be maintained through the following processes:

- **Regular Updates:**
  - Test cases updated with new features
  - Bug fixes and regression testing
  - Documentation updates in GitHub
- **Code Quality:**
  - ESLint for code style enforcement
  - TypeScript for type safety
  - Code review process for new features
- **Documentation:**
  - README updates for setup instructions
  - API documentation
  - Test case documentation

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?

   There was a lot of effective communication and problem solving throughout the development of the VnV plan. The team was able to develop clear sections and specific testing categories for features such as the authentication, access control, game scheduling, etc. This made is easier to organize the structure of the document. Using a google doc, the team was able to create an intial draft with tasks assigned to team members either individually, or in sub groups. Following the creation of the ideas brainstormed in the draft, we created a formal documentation of the VnV plan utilizing LaTeX.

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

   A challenge we faced during the formulation of this deliverable was ensuring the traceability between test cases and requirements listed in section 4.4. The process of tracking each requirements to their respective test cases can easily become complex. To resolve this issue, we established a mapping method to simplify the traceability.

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.

Static and dynamic testing knowledge will be an important skill the team will collectively need for the completion of verification and validation. Thiw will be critical for validating both functional and non functional requirements.

Usability and performance testing will be a necessary skill for the verification and validation, as it will be essential for the understanding of the human factors and performance for the non functional requirements.

Security testing will be an important topic for the completion of this deliverable due to the teams emphasis on security. A proper understanding of cybersecurity principles will help improve the security.

Tool proficiency will a skill necessary for the team to acquire to improve the automated testing tools, as well as the efficiency of and accuracy of the testing.

Traceability and Requirements Management will be crucial for our team to understand how to link each requirement to test cases specific test cases.

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?

   Static and dynamic testing knowledge - Damien

   Damien will work on enhancing his statis and dynamic testing knowledge with the assitance of online resources.

   (a) Taking an online course with a focus on software testing methods.

   (b) Organize a peer learning session where team members individually search and share various testing techniques.

   Usability and performance testing - Emma

   Emma will work on her usability and performance testing skills to improve the teams performance metrics.

(a) Evaluate our perfomance matrix using analytical tools.

(b) Learn more about UX/UI principles using online resources.

Security testing - Tuoyo

Tuoyo will focus on improving his skills with the security testing to assist the team with improving security practices.

(a) Enroll in a cybersecurity course to learn core concepts.

(b) Learn more about common vulnerablities and techniques to mitigate them

Tool proficiency - Jad

Jad will work on tool proficiency to improve the teams automated processes.

(a) Practice using selected tools in examples scenarios.

(b) Attend vendor webinars or tutorials for selected tools.

Traceability and Requirements Management - Derek

Derek will work on the Traceability and Requirements Management to improve the validation and verification of requirements.

(a) Learn the use of requirement management tools to facilitate traceability.

(b) Research and study best practices in requirement management.

Notes

This section will be completed at the end of Phase 2 as specified in section 21.2.2 in the SRS.

9 Appendix

This is where you can place additional information.

9.1 Symbolic Parameters

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

parameter	value	unit	description
MIN_NAVTIME	60	s	the minimum time for testers to navigate to a main view
MIN_TESTERS	5	n/a	the minimum testers required for a certain system test
AVG_TASK_TIME	3	min	the average task time for a task completion
MAX_GAME_DIFF	2	n/a	the maximum difference of total scheduled games between two teams

Table 5: Symbolic Parameters

9.2 Usability Survey Questions

Learning

- How easy was it to learn how to use the platform? (1-5)
- How helpful were the instructions or help resources in learning how to use the platform? (1-5)
- How confident do you feel using the platform without assistance after your initial experience? (1-5)

Usability

- How intuitive do you find the navigation of the platform? (1-5)
- Did you encounter any errors or issues while using the platform? If so, please describe them. (Open-ended)

Accessibility

- How accessible do you find the platform in terms of visual design (e.g., color contrast, text size, font)? (1-5)
- Did you encounter any errors or issues while using the platform? If so, please describe them. (Open-ended)
- Do you have any suggestions for improving the accessibility of the platform? (Open-ended)