Unit Test Plan

Product Owner Responsibilities

1. Reviews pull requests to ensure no bugs are pushed into the main branch.

2. Maintains the foundation of what needs to be developed, especially around database interactions and UI display.

3. Confirms when a feature (e.g., stadium display, CSV import) has been correctly implemented.

4. Manages feature prioritization and sprint planning.

5. Answers any team questions about feature requirements or design decisions.

Scrum Master Responsibilities

1. Keeps a log of each scrum meeting.

2. Ensures the team is working efficiently and encourages continuous improvement.

3. Tracks progress and ensures team members don't overcommit.

4. Helps the Product Owner review pull requests if needed.

5. Ensures code follows standards and adheres to scrum practices.

Other Team Member Responsibilities

- Developers implement features, write Qt unit tests, and run desk checks.

- Testers validate feature functionality and report bugs.

- All members collaborate on bug fixing and review.

Testing Approach

- **Testing Type:** Both **black box** (UI output verification) and **white box** (logic and query validation) testing are used.

- **Desk Checks:** Developers check for prominent issues in code before pushing.

GitHub Workflow

- **Branching Structure:**
    - `main`: Stable production code.
    - `dev`: Integration branch.
    - `main-*`: One branch per feature (e.g., `main-csv-import`, `main-display-sorting`).
- **Process:**
    - Developers commit to feature branches, open pull requests into `dev`, and after testing, merge into `main`.

Story Completion and Testing

- When a story is completed, a pull request is submitted and reviewed by the Product Owner or Scrum Master.
- Unit tests are executed using the Qt Test module.
- If a story fails testing, team members swarm to fix it—bugs are not simply pushed to the backlog unless necessary.

Environment

- **Operating Systems:** Windows (primary) and macOS (some users).
- **Software:**
    - Qt Creator with native Qt on all platforms
    - SQLite for database (including a small test database)
    - GitHub for version control
    - CSV files for testing import logic

Testing Details

Database

- A small SQLite test database (~5–10 teams) is used for all tests.

Display Functions (All tested independently):

- Display teams sorted by team name
- Display teams sorted by stadium name
- Display American League teams only
- Display National League teams only
- Display teams grouped/sorted by typology
- Display teams with open roofs
- Display teams by stadium opening date
- Display teams by seating capacity
- Display team with greatest center field size
- Display team with smallest center field size

CSV Import:

- Valid CSV file import — verify correct data insertion using `loadStadiumMap()` behavior
- Empty CSV file — confirm no crash, no data added
- Malformed CSV — errors are handled gracefully (rows skipped or rejected without crashing)

Error Handling:

- Display behavior when the database is disconnected
- Display behavior when queries return no results (UI should show empty table or message)

Deliverables

- Correct implementation of display and import features
- Unit tests using Qt Test and QSignalSpy
- Error handling and UI stability
- Working test database

Glossary

- **Black Box Testing** – Tests based on inputs and outputs without seeing internal logic.
- **White Box Testing** – Tests based on knowing the internal workings of the code.
- **Configuration Management** – Version control, branching, and pull request practices using GitHub.
- **QSignalSpy** – Qt utility to confirm signal-slot communication.
- **Desk Check** – Informal code review before formal testing.

Features Not Tested

- GUI layout, colors, fonts
- SQLite internal reliability (assumed to be correct)
- Initial database construction outside of team logic