MSSE SOFTWARE, INC.

Test Plan for

GolfScore

Contents

1.0 Introduction

1.1. Objective

This document describes the test plan for the GolfScore application. The main purpose of this testing is to verify the compliance of the application with the requirements specified in the Software Requirements Specification (SRS). The plan includes information on what will be tested, how the testing will be performed (testing methodology), schedule, resources, entry and exit criteria, dependencies, testing tools, metrics, and testing requirements.Project Description

1.2. Project Description

The GolfScore project is a software application designed to process and generate reports from golf tournament data. The primary function of the application is to take input data, which includes details about golf courses and player scores, and produce comprehensive reports that rank players based on their performance across multiple courses. The application operates via a command-line interface (CLI) and is intended to be a stand-alone executable that can run on a PC with Windows 2000 or later.

The key outputs of the application are:

- **Tournament Ranking Report:** A list of all golfers with their scores for each course and their total tournament score, ranked from highest to lowest.
- **Golfer Report:** Similar to the Tournament Ranking Report, but sorted alphabetically by the golfers' last names.
- **Course Report:** For each course, a report showing the golfers' hole-by-hole scores and total score, sorted by performance on that course.

1.3. Process Tailoring

This project will use standard software development and management processes as a guideline. However, due to the specific requirements and constraints of the GolfScore project, some modifications to these processes will be necessary. The following adjustments will be made:

- **Testing Types:**
    - **Specification Testing:** To verify that the GolfScore application meets the specifications outlined in the Software Requirements Specification (SRS).
    - **Functional Testing:** To ensure that the program functions correctly according to the defined requirements, such as correctly processing input data and generating the required reports.

- **Limits Testing:** To determine the application's behavior under conditions of maximum and minimum inputs, such as testing with the maximum number of golfers and courses.
- **Stress Testing:** To evaluate the application's performance under high load conditions, such as processing a large number of input files simultaneously.
- **Error Handling Testing:** To verify that the application correctly handles and reports errors, including invalid data and incorrect command-line parameters.
- **Compatibility Testing:** Since the application is intended to run on Windows 2000 or later, compatibility testing will be conducted to ensure it functions correctly across different versions of Windows operating systems.

1.4. Referenced Documents:

- **Software Requirements Specification (SRS):** GolfScore SRS, Revision 1.1, July 18, 2017.
- **Test Plan Example:** Document used as a reference for structuring the test plan and procedures.
- **Software Development Plan:** Although not specifically detailed in the provided documents, standard development practices will be applied in accordance with industry standards.

2.0 Assumptions/Dependencies

Assumptions:

- The GolfScore application is fully developed and feature-complete before testing begins. This includes the successful integration of all modules and functions as described in the Software Requirements Specification (SRS).
- The input files used for testing are correctly formatted according to the specifications outlined in the SRS. These files will include valid and invalid data to test the application's ability to handle different scenarios.
- The test environment will be stable and will include access to a PC running Windows 2000 or later, which is necessary to execute the GolfScore application.
- The development team will be available for consultation and to address any defects or issues identified during testing promptly.

Dependencies:

- **Code Completion:** The testing schedule is dependent on the timely completion of the development phase. The code must be complete, and a stable version of the GolfScore application must be provided by the development team by [insert expected date here] to allow sufficient time for testing.

- **Prototype Availability:** A fully functional prototype of the application should be available for testing. This prototype should include all features that will be present in the final release.
- **Integration and System Verification Testing (SVT):** It is assumed that unit and integration tests have already been conducted by the development team before the start of formal testing. The results of these tests should be provided to the testing team.
- **External Resources:** Access to necessary testing tools, such as text editors and file comparison tools, should be ensured. Additionally, any third-party tools or libraries used in the development of GolfScore should be stable and functioning as expected.

3.0 Test Requirements

3.1 Functional Requirements:

1. **Program Invocation:**
    - The program must correctly process command-line parameters, including displaying help (-h), generating reports (-c, -t, -g), and handling input files and output directories.
2. **Tournament Data Processing:** The program must correctly process the input file containing records for golf courses and player results. The following parameters must be considered:
    - The number of courses can range from 1 to 5.
    - The number of players can range from 2 to 12.
    - The par for each hole must be 3, 4, or 5 strokes.
    - Correct calculation of scores for each hole and the total score for each player.
3. **Report Generation:** The program must generate up to three types of reports in the corresponding text files:
    - **Tournament Report:** A list of all players with their scores for each course and the total score, sorted in descending order.
    - **Player Report:** Similar to the Tournament Report, but sorted alphabetically by the players' last names.
    - **Course Report:** For each course, a list of players with hole-by-hole scores and the total score, sorted in descending order.
4. **Error Handling:** The program must correctly handle errors in the input data:
    - Incorrect command-line parameters should result in an appropriate error message.

- Incorrect data in the input file (e.g., non-numeric values where numeric values are expected) should cause the program to stop and display an appropriate error message.
- If a player has multiple records for the same course, the program should ignore duplicate records after the first one and display an error message.

5. **Output File Handling:**
    - The program must correctly handle situations where an output file already exists. The user should be prompted to confirm the overwriting of the file.
    - If the specified output file does not exist, the program should create it automatically.

3.2 Performance Requirements:

1. **Performance Requirements:**
    - The program must complete the processing of data and generation of reports within one minute after execution.

4.0 Test Tools

Existing Tools:

1. **Text Editors:**
    - Tools such as Notepad++ or Visual Studio Code will be used to create and edit the input files needed for testing the GolfScore application.
2. **Command-Line Interface (CLI):**
    - The Windows command prompt will be used to execute the GolfScore application and observe its behavior with various input parameters.
3. **File Comparison Tools:**
    - Tools like diff or WinMerge will be utilized to compare the generated output reports with expected results to verify correctness.

Tools to be Developed or Purchased:

1. **Automated Testing Scripts:**
    - Depending on the complexity of the test cases, it may be necessary to develop custom scripts that automate the execution of the GolfScore application with different input scenarios. These scripts can be developed using languages like Python or batch scripting.

- The development process will include writing, testing, and refining these scripts to ensure they accurately simulate the different test scenarios.

2. **Test Management Software:**
    - If not already available, a test management tool such as Jira or TestRail might be purchased to track test cases, defects, and the overall progress of the testing process.

5.0 Resource Requirements

Human Resources:

- **Test Engineer:**
    - One dedicated test engineer will be required to design, execute, and document the test cases. The engineer will also be responsible for developing any necessary test scripts, managing test data, and tracking defects.
    - Time Commitment: Full-time during the test execution phase.
- **Development Team Support:**
    - Availability of a developer or development team for consultation during the testing phase. This resource will be necessary for resolving any defects or issues that arise during testing.
    - Time Commitment: Part-time, as needed.

Hardware Resources:

- **Test Environment:**
    - A PC running Windows 2000 or later, configured with the necessary software tools such as text editors, command-line interface, and file comparison tools.
    - Backup or additional systems may be required to simulate different environments or to handle parallel testing efforts.

Software Resources:

- **Test Management Software:**
    - If not already available, a test management tool (e.g., Jira, TestRail) should be purchased or licensed to track test cases, manage defects, and monitor the progress of testing.

- **Automated Testing Tools:**
    - Development or acquisition of automated test scripts to streamline the execution of repetitive test cases. These scripts will be developed using scripting languages like Python or shell scripting.

Time Resources:

- **Test Planning and Design:**
    - Estimated Time: 1 week
- **Test Execution:**
    - Estimated Time: 2 weeks
- **Test Reporting and Closure:**
    - Estimated Time: 1 week

6.0 Test Schedule

Test Planning and Preparation:

- **Start Date:** August 14, 2024
- **End Date:** August 20, 2024
- **Activities:**
    - Finalize the test plan and test cases.
    - Develop any necessary test scripts and set up the test environment.
    - Prepare input data and configure tools.

Entrance Testing:

- **Start Date:** August 21, 2024
- **End Date:** August 22, 2024
- **Activities:**
    - Execute initial tests to verify that the application is ready for full testing.
    - Identify and resolve any immediate issues that could impede further testing.

Main Testing:

- **Start Date:** August 23, 2024
- **End Date:** August 30, 2024
- **Activities:**
    - Execute all planned test cases, including functional, performance, and error-handling tests.
    - Document test results and identify any defects.

Regression Testing:

- **Start Date:** September 1, 2024
- **End Date:** September 3, 2024
- **Activities:**
    - Re-test the application after defects have been fixed to ensure no new issues have been introduced.
    - Validate that all previous issues have been resolved.

Test Reporting and Closure:

- **Start Date:** September 4, 2024
- **End Date:** September 6, 2024
- **Activities:**
    - Compile the final test report, summarizing all findings, defects, and their resolutions.
    - Conduct a final review meeting with the Project Manager and Development Team to confirm that the application is ready for release.

7.0 Risks/Mitigation

1. **Delays in Code Completion:**
    - **Risk:** The development team may not deliver the final version of the code on schedule, which could delay the start of testing.
    - **Mitigation:** Maintain close communication with the development team to monitor progress and adjust the testing schedule if necessary.

2. **Inadequate Test Data:**
    - **Risk:** The test data may not be sufficiently comprehensive or may be incorrectly formatted, leading to incomplete testing coverage.
    - **Mitigation:** Ensure that the test data is prepared and reviewed early in the test planning phase. Create multiple scenarios to cover all possible cases.
3. **Limited Resource Availability:**
    - **Risk:** The test engineer or development team may be unavailable during critical phases of testing, leading to delays.
    - **Mitigation:** Schedule regular check-ins and ensure that backup resources are identified in advance in case of unavailability.
4. **Defects in the Application:**
    - **Risk:** Critical defects in the application may be discovered late in the testing phase, requiring significant rework and additional testing.
    - **Mitigation:** Prioritize testing of critical functions early in the process and conduct thorough regression testing after fixes are applied.
5. **Tool or Environment Failures:**
    - **Risk:** Issues with testing tools or the test environment could disrupt testing activities.
    - **Mitigation:** Have backup tools and environments ready and test the tools before the main testing phase begins.

8.0 Metrics

Prior to Shipment:

1. **Effort Expended:**
    - Track the amount of time and resources spent during Development Verification Testing (DVT), System Verification Testing (SVT), and Regression Testing.
2. **Number of Defects Uncovered:**
    - Record the total number of defects identified during DVT, SVT, and Regression Testing. Additionally, categorize each defect by the development phase in which it was introduced.
3. **Test Tracking S-Curve:**

- Use an S-Curve to monitor the progress of test case execution over time, providing a visual representation of the testing process.

4. **Problem Tracking and Resolution (PTR) S-Curve:**
    - Track the discovery and resolution of defects using a PTR S-Curve to visualize the rate at which defects are being identified and resolved.

After Shipment:

1. **Number of Defects Uncovered:**
    - Continue to track the number of defects discovered after the product has been shipped. Categorize these defects by the phase of development in which they were introduced.
2. **Size of Software:**
    - Measure and document the size of the software in terms of lines of code or another appropriate metric, providing context for the number of defects relative to the software's complexity.

Appendix A – Detailed Resource Requirements

Test Activities and Estimated Hours:

1. **Test Planning and Design:**
    - **Activity:** Developing the test plan, designing test cases, and setting up the test environment.
    - **Estimated Hours:** 40 hours
    - **Responsible Engineer:** [Engineer Name]
2. **Development of Test Scripts:**
    - **Activity:** Writing and testing automated scripts to execute test cases.
    - **Estimated Hours:** 30 hours
    - **Responsible Engineer:** [Engineer Name]
3. **Entrance Testing:**
    - **Activity:** Conducting initial tests to ensure the application is ready for full testing.
    - **Estimated Hours:** 16 hours
    - **Responsible Engineer:** [Engineer Name]
4. **Main Testing:**
    - **Activity:** Executing all functional, performance, and error-handling tests.
    - **Estimated Hours:** 80 hours
    - **Responsible Engineer:** [Engineer Name]
5. **Regression Testing:**
    - **Activity:** Re-testing the application after fixes to ensure no new issues have been introduced.
    - **Estimated Hours:** 24 hours
    - **Responsible Engineer:** [Engineer Name]
6. **Test Reporting:**
    - **Activity:** Documenting test results, compiling final reports, and conducting review meetings.
    - **Estimated Hours:** 20 hours

- **Responsible Engineer:** [Engineer Name]

1.1.1.1. Grand Total of Effort:

- **Total Estimated Hours:** 210 hours

Appendix B – Test Cases

The description of each test case will include the following:

- **Test Name**
- **Description**
- **Execution Steps**
- **Expected Result**

Test Case Examples:

Test Case 1:
o **Test Name:** Basic Valid Input
o **Description:** Verify the application processes a valid input file containing 3 golf courses and 4 players.
o **Execution Steps:**
1. Provide a valid input file with 3 courses and 4 players.
2. Execute the application.
o **Expected Result:** The application successfully generates all three reports.
Test Case 2:
o **Test Name:** Invalid Score Data
o **Description:** Verify the application handles invalid par values for some holes.
o **Execution Steps:**
1. Provide an input file where some par values are invalid (not 3, 4, or 5).
2. Execute the application.
o **Expected Result:** The application should stop and display an error message.
Test Case 3:
o **Test Name:** Incorrect Command-Line Parameters
o **Description:** Verify the application handles incorrectly formatted

command-line parameters.
○ **Execution Steps:**
1. Execute the application with incorrect command-line parameters.
○ **Expected Result:** The application displays an error message.
Test Case 4:
○ **Test Name:** Minimum Number of Golf Courses and Players
○ **Description:** Verify the application processes a file with the minimum number of courses and players.
○ **Execution Steps:**
1. Provide an input file with 1 course and 2 players.
2. Execute the application.
○ **Expected Result:** The application successfully generates reports without errors.
Test Case 5:
○ **Test Name:** Maximum Number of Golf Courses and Players
○ **Description:** Verify the application processes a file with the maximum number of courses and players.
○ **Execution Steps:**
1. Provide an input file with 5 courses and 12 players.
2. Execute the application.
○ **Expected Result:** The application successfully generates reports without errors.
Test Case 6:
○ **Test Name:** Duplicate Player Records
○ **Description:** Verify the application handles duplicate player records for the same course.
○ **Execution Steps:**
1. Provide an input file where one player has two records for the same course.

2. Execute the application.
○ **Expected Result:** The application ignores the duplicate record, displays an error message, and continues processing.
Test Case 7:
○ **Test Name:** Help Command
○ **Description:** Verify the application displays help information when executed with the -h parameter.
○ **Execution Steps:**
1. Execute the application with the -h parameter.
○ **Expected Result:** The application displays help information on the screen.
Test Case 8:
○ **Test Name:** Creating Report in Non-Existent Directory
○ **Description:** Verify the application creates output directories if they do not exist.
○ **Execution Steps:**
1. Execute the application with a valid input file and specify a non-existent output directory.
○ **Expected Result:** The application creates the directory automatically and saves the reports without errors.
Test Case 9:
○ **Test Name:** Overwriting Existing Report
○ **Description:** Verify the application asks for confirmation before overwriting an existing report.
○ **Execution Steps:**
▪ Execute the application with parameters that generate a report in an existing file.
▪ Confirm the overwrite.
○ **Expected Result:** The application prompts for confirmation and overwrites the file if confirmed.

Test Case 10: Invalid Command-Line Option
○ **Test Name:** Handling of Invalid Command-Line Options
○ **Description:** Verify that the program correctly handles the input of an invalid or unsupported command-line option.
○ **Execution Steps:**
▪ Execute the program with an unsupported command-line option (e.g., `-z`).
○ **Expected Result:** The program displays an error message indicating that the option is not supported and provides a list of valid options.
Test Case 11: Empty Input File
○ **Test Name:** Handling of Empty Input File
○ **Description:** Verify that the program correctly handles an empty input file without crashing.
○ **Execution Steps:**
▪ Provide an empty input file and execute the program.
○ **Expected Result:** The program should display an error message indicating that the input file is empty and terminate without generating any reports.
Test Case 12: Large Number of Golfers
○ **Test Name:** Handling of Large Number of Golfers
○ **Description:** Verify the program's behavior when the number of golfers exceeds the maximum limit (e.g., 13 golfers).
○ **Execution Steps:**
▪ Provide an input file with 13 golfers and execute the program.
○ **Expected Result:** The program should display an error message indicating that the maximum number of golfers is exceeded and not proceed with processing.
Test Case 13: Malformed Input File
○ **Test Name:** Handling of Malformed Input File
○ **Description:** Verify that the program correctly handles input files with

missing or misaligned data.
○ **Execution Steps:**
▪ Provide an input file where some data fields are missing or columns are misaligned.
○ **Expected Result:** The program should display an appropriate error message and stop further processing.
Test Case 14: Non-Numeric Stroke Count
○ **Test Name:** Handling of Non-Numeric Stroke Count
○ **Description:** Verify the program's ability to handle cases where stroke counts are non-numeric characters.
○ **Execution Steps:**
▪ Provide an input file where the stroke count for one or more holes is non-numeric (e.g., "A" instead of "4").
○ **Expected Result:** The program should display an error message indicating invalid data and terminate processing.
1.1.1.2. Non-Functional Requirements:
Test Case 15: Security - Unauthorized Access to Output Directory
○ **Test Name:** Unauthorized Access to Output Directory
○ **Description:** Verify that the program correctly handles attempts to write output files to a directory where the user does not have write permissions.
○ **Execution Steps:**
▪ Execute the program and specify an output directory where the current user does not have write permissions.
○ **Expected Result:** The program should display an error message indicating that it does not have the necessary permissions to write to the specified directory and terminate gracefully.
Test Case 16: Usability - Help Command Format
○ **Test Name:** Format and Clarity of Help Command Output
○ **Description:** Verify that the help command output is clear, concise, and provides all necessary information to the user.

○ **Execution Steps:**
▪ Execute the program with the `-h` command-line option.
○ **Expected Result:** The program should display help information in a clear and organized format, including descriptions of all valid options and usage examples.
Test Case 17: Performance Under Load
○ **Test Name:** Performance Under Heavy Load
○ **Description:** Verify the program's performance when processing a very large input file (e.g., maximum number of golfers and courses with extensive data).
○ **Execution Steps:**
▪ Provide a large input file and execute the program.
○ **Expected Result:** The program should complete processing within the specified performance limits (e.g., within one minute), without crashing or significantly slowing down.
Test Case 18: Recovery from Unexpected Shutdown
○ **Test Name:** Recovery from Unexpected Shutdown
○ **Description:** Verify how the program handles recovery after an unexpected shutdown (e.g., power failure) during processing.
○ **Execution Steps:**
▪ Simulate an unexpected shutdown during the processing of an input file.
▪ Restart the program and reattempt the processing.
○ **Expected Result:** The program should handle the unexpected shutdown gracefully and allow the user to resume or restart the processing without data corruption.