MSSE SOFTWARE, INC.

Test Plan for GolfScore

Revision 1.1

January 4, 2024

Mwangi Anthony Maina

Contents

1.0 Introduction

1.1. Objective

This Test Plan is an aggregation of information, which describes the entire test activity for this project. It covers the entire testing effort (unit, development test, system verification test, and Beta). It identifies the product requirements, schedules, resource requirements (people, effort and equipment), quality, assumptions, exclusions, and risks.

A preliminary Test Plan is prepared for the Project Team during the System Phase of PEAQ Process. This Test Plan will be updated in the earliest possible time of the Implementation Phase, so that progress can be tracked during implementation.

1.2. Project Description

This project is about developing a GolfScore program used to generate reports of golfers' results for a golf tournament. The program will be a stand alone executable only run through a command line interface.

The input to the program will consist of a file containing course records and golfer records. The output from the program will consist of up to 3 reports which are tournament ranking record, golfer record and course record.

1.3. Process Tailoring

1. Test Strategy:

Adaptation to Command Line Interface (CLI):

Given that GolfScore is executed via a command line interface and has no GUI, the testing focus will be on command-line input validation, accurate processing, and correct generation of reports.

Integration of Unit and System Testing:

Due to the standalone nature of the application, there will be a strong emphasis on integration testing along with unit testing to ensure that individual components work well together.

2. Types of Testing:

Functional Testing:

- Verify correct interpretation of command-line options.
- Validate the generation of Course, Tournament Ranking, and Golfer reports.
- Check handling of errors in input data and parameters.

Limits and Stress Testing:

- Test the application with the maximum allowed number of golf courses (5), golfers (12), and holes per course (18).
- Stress the application with large data sets to assess its performance under heavy loads.

Documentation Testing:

- Confirm that the generated reports match the specified format in the SRS.
- Verify that help information is correctly displayed when the -h option is used.

3. Testing Scenarios:

Command-Line Options:

- Test various combinations of command-line options (e.g., -c, -t, -g, -h) to ensure correct behavior.
- Check for proper handling of invalid options.

File Input:

- Test with valid input files containing Course and Golfer records.
- Validate error handling for non-existent input files.

File Output:

- Verify the creation of the specified output files and their content.
- Test the overwrite prompt functionality.

Data Validation:

- Ensure proper validation of numeric data, par values, and duplicate golfer records.

Performance Testing:

- Assess the execution time for the worst-case scenario with maximum allowed data.

4. Rationale for Tailoring:

- Limited GUI Testing: Since the application does not have a graphical user interface, GUI testing steps are not applicable.
- Streamlined Documentation Verification: The focus will be on ensuring that the generated reports match the specified format without extensive documentation testing.
- No Compatibility Testing: Compatibility testing will not be necessary as the application is designed to run on a PC with Windows 2000 or later.

Referenced Documents

1. Software Requirements Specification for GolfScore Rev. 1.1, July 18, 2017

2.0 Assumptions/Dependencies

1. Code Completeness:

It is assumed that the GolfScore executable file will be available for testing with all the specified functionalities implemented according to the SRS document.

2. Responsibility for Unit and Integration Tests:
The development team is responsible for conducting unit and integration tests to ensure individual components and their interactions are functioning correctly.

3. Availability of Test Data:

Test data, including sample input files and expected output files, will be provided and accessible for conducting functional and scenario-based testing.

4. Stable testing Environment:

The testing environment, including the operating system (Windows 2000 or later) and any necessary dependencies, will remain stable throughout the testing period.

5. Timely Issue Resolution:

Timely resolution of any issues or defects identified during testing is crucial for maintaining the project schedule.

3.0 Entry Criteria

- Availability of SRS Document
- Test Environment Setup
- Availability of Test Resources
- Completion of Unit and Integration Testing
- Code Freeze
- Availability of Test Tools
- Traceability Matrix
- Test Plan Approval

4.0 Exit Criteria

- Test Execution Completion
- Defect Resolution
- Test Reports Generated
- Code Coverage Analysis
- Performance Metrics
- Validation of Requirements
- Stakeholder Approval
- Documentation Review
- Knowledge Transfer
- Test Environment Cleanup
- Training and Skill Enhancement
- Final Project Closure

5.0 Test Requirements

1. Command-Line Interface (CLI) Testing:

1.1 Verify the correct interpretation of command-line options (-h, -c, -t, -g).

1.2 Validate that multiple options can be combined (e.g., -cg).

1.3 Confirm the accurate handling of invalid options, displaying appropriate error messages.

2. File Input and Output Testing:

2.1 Test with valid input files containing Course and Golfer records.

2.2 Validate error handling for non-existent input files.

2.3 Verify that the generated reports match the specified format in the SRS.

2.4 Confirm the creation of specified output files and their content.

2.5 Test the overwrite prompt functionality when generating existing reports.

3. Data Validation Testing:

3.1 Ensure proper validation of numeric data where numeric data is expected.

3.2 Validate par values for holes, ensuring they are 3, 4, or 5.

3.3 Check for duplicate golfer records for the same golf course, ignoring additional records after the first and displaying a message.

4. Functional Testing:

4.1 Test the generation of Course, Tournament Ranking, and Golfer reports.

4.2 Validate the content of the Tournament Ranking Report:

4.3 Validate the content of the Golfer Report, listing golfers alphabetically by last name.

4.4 Validate the content of the Course Report for each golf course specified in the input Course Records:

5. Limits and Stress Testing:

5.1 Test the application with the maximum allowed number of golf courses (5), golfers (12), and holes per course (18).

5.2 Stress the application with large data sets to assess its performance under heavy loads.

6. Error Handling Testing:

6.1 Verify that input parameter errors, including unrecognizable options, result in the program stopping and displaying appropriate error messages.

6.2 Validate input parameter errors for non-existent input files and directories, displaying relevant error messages.

6.3 Test error handling for non-numeric data where numeric data is expected.

6.4 Confirm error handling for par values that are not 3, 4, or 5.

6.5 Check for user prompts for overwriting existing reports.

7. Performance Testing:

7.1 Assess the execution time for the worst-case scenario with the maximum allowed data.

6.0 Test Tools

- Command-Line Testing Framework
- Custom Test Scripts or File Comparison Tools
- Data Validation Tools
- Test Automation Framework
- Performance Testing Tools (e.g., Apache JMeter, Gatling)
- Error Injection Tools
- Test Management Tools (e.g., TestRail, Zephyr)
- Profiling Tools, Load Testing Tools

7.0 Resource Requirements

1. Test Team:

1.1 Test Manager

1.2 Test Lead

1.3 Test Engineers

2. Testing Environment:

2.1 Test Machines:

Machines for executing tests on different operating systems (Windows 2000 or later).

2.2 Software:

Development environment for running unit and integration tests.

2.3 Test Data:

Sample input files with Course and Golfer records.

3. Documentation:

3.1 Test Plan Documenter

3.2 Test Report Documenter

5. Infrastructure:

5.1 Test Servers:

Servers for hosting tools and test management systems.

8.0 Test Schedule

We will be done 15 weeks after the Unit and Integration testing conducted by the developers team.

1. Test Planning and Preparation:

Duration: 1 Week

Activities:

- Review the Software Requirements Specification (SRS) document.
- Identify and analyze test requirements.
- Develop the detailed test plan.
- Define test scenarios and test cases.
- Allocate resources and establish roles.
- Set up the testing environment.

2. Unit and Integration Testing:

Duration: 2 Weeks

Activities:

- Developers conduct unit tests.
- Integration tests to ensure proper collaboration between components.

3. Command-Line Testing:

Duration: 1 Week

Activities:

- Execute command-line interface (CLI) tests.
- Validate correct interpretation of options.
- Verify handling of invalid options.

4. File Input and Output Testing:

Duration: 2 Weeks

Activities:

- Automated testing of file input and output scenarios.
- Validate report generation and correctness.
- Test overwrite prompt functionality.

5. Data Validation Testing:

Duration: 1.5 Weeks

Activities:

- Automated testing of data validation scenarios.
- Check numeric data validation and par values.
- Validate handling of duplicate golfer records.

6. Functional Testing:

Duration: 3 Weeks

Activities:

- Execute functional tests to validate report content.
- Validate Tournament Ranking, Golfer, and Course reports.
- Address any issues identified during testing.

7. Limits and Stress Testing:

Duration: 2 Weeks

Activities:

- Test application with maximum allowed data.
- Stress testing to assess performance under heavy loads.
- Address any performance bottlenecks.

8. Error Handling Testing:

Duration: 1.5 Weeks

Activities:

- Automated testing of various error scenarios.
- Validate error messages and prompt functionality.

9. Performance Testing:

Duration: 2 Weeks

Activities:

- Assess execution time for worst-case scenarios.
- Evaluate resource utilization and identify optimizations.

10. Test Reporting and Documentation:

Duration: 1 Week

Activities:

- Document test execution results.
- Generate and finalize test reports.
- Log and track defects.

11. Review and Sign-off:

Duration: 1 Week

Activities:

- Review test results and reports.
- Conduct a final review meeting.
- Obtain stakeholders' sign-off.

9.0 Risks/Mitigation

1. Risk: Incomplete or Ambiguous Requirements:

Mitigation: Conduct thorough reviews and clarification sessions with stakeholders to ensure a comprehensive understanding of requirements. Regularly update the test plan as requirements evolve.

2. Risk: Unstable Development Environment:

Mitigation: Collaborate closely with the development team to stay informed about code changes. Establish version control and conduct frequent integration testing to identify and address integration issues early.

3. Risk: Limited Test Data:

Mitigation: Generate diverse and realistic test data sets to cover various scenarios. Develop scripts to automate data creation when possible. Collaborate with the development team to obtain or create representative data.

4. Risk: Insufficient Test Coverage:

Mitigation: Create a traceability matrix to map test cases to requirements. Regularly review and update test cases to ensure comprehensive coverage. Leverage code coverage tools to identify areas that need additional testing.

5. Risk: Delayed Delivery of Test Environments:

Mitigation: Establish communication channels with the IT and infrastructure teams to prioritize test environment setup. Identify alternative environments or cloud-based solutions to reduce dependencies.

6. Risk: Changes in Test Schedule:

Mitigation: Regularly monitor project progress and adjust the test schedule accordingly. Maintain open communication with stakeholders and provide early notifications of potential delays.

7. Risk: Inadequate Performance under Load:

Mitigation: Perform thorough stress testing to identify performance bottlenecks. Optimize code and database queries as needed. Monitor resource utilization during testing to ensure scalability.

8. Risk: Insufficient Time for Test Execution:

Mitigation: Prioritize test cases based on critical functionality. Automate repetitive and time-consuming tests. Continuously monitor test progress and adjust the test schedule as needed.

9. Risk: Overwriting Existing Reports:

Mitigation: Implement confirmation prompts for overwriting existing reports. Clearly communicate potential risks to users through system messages.

10.0 Metrics

1. Test Coverage Metrics:

Measure the percentage of requirements covered by executed test cases.

2. Defect Density:

Measure the number of defects identified per unit of code.

3. Test Execution Progress:

Track the percentage of test cases executed over time.

4. Test Automation Coverage:

Measure the percentage of test cases automated.

5. Defect Closure Rate:

Measure the rate at which reported defects are resolved and closed.

6. Test Pass Rate:

Measure the percentage of test cases passing successfully.

7. Performance Test Metrics:

Definition: Measure key performance indicators, such as response times and resource utilization, under various load conditions.

8. Test Environment Stability:

Measure the availability and stability of the test environment.

9. Regression Test Suite Effectiveness:

Measure the effectiveness of the regression test suite in identifying new defects.

10. Test Documentation Completeness:

Assess the completeness of test documentation, including test plans, test cases, and test reports.

11. Resource Utilization:

Evaluate the efficient use of testing resources, including testers, tools, and equipment.

Appendix A – Test cases

1.	The number of golf course '1' shall be accepted	Functional
2.	The number of golf course '5' shall be accepted	Functional
3.	The number of golf course '6' shall return an error	Functional
4.	The number of golf course '0' shall return an error	Functional
5.	The number of the golfer’s stroke count 'A' shall return an error	Functional
6.	par value for course hole ‘2’ shall return an error	Functional
7.	Command line option ‘ -,c’ shall return an error	Functional