MSSE SOFTWARE, INC.

Test Plan for GolfScore

Revision P1

Valentyn Harkovenko

February 21, 2023

Contents

1. Introduction

1.1. Objective

The 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. Test Description

The GolfScore program is designed to read a data file that contains information about golfers, their scores, and other details. It then processes this data to generate different reports based on user input. The program has several functional and non-functional requirements that need to be tested.

The tests planned for this project are as follows:

- Specification testing: This testing will ensure that the program meets all its functional and non-functional requirements as specified in the Software Requirements Specification (SRS).
- Functional testing: This testing will verify that the program performs all the functions as expected. It will cover testing of all input parameters, error handling, data processing, and output generation.
- Limits testing: This testing will verify that the program can handle large data files and generate reports for a large number of golfers.
- Compatibility testing: This testing will verify that the program works correctly on different platforms and with different versions of operating systems.
- Performance testing: This testing will verify that the program can process data and generate reports within one minute as specified in the requirements.
- Documentation testing: This testing will ensure that the program documentation is accurate, complete, and understandable.

1.3. Process Tailoring

The GolfScore program development and management process will be based on standard software development and management processes. However, some tailoring of the process based on unique project requirements will be made. The following process tailoring will be used:

- The testing will be done in parallel with the development process to ensure that any issues are detected and fixed as soon as possible.
- The testing team will work closely with the development team to ensure that the program meets all the requirements and specifications.

1.4. Referenced Documents

The following documents were used to produce this Test Plan:

- Software Requirements Specification (SRS)

2. Assumptions/Dependencies

Assumptions:

- The software is developed according to the specified requirements.
- The necessary hardware and software are available for testing.
- The test team has access to the necessary resources for testing.
- The test team has the required skills and expertise for testing.
- The test team has access to the test environment.

Dependencies:

- Completion of code development by the specified date to meet the test schedule.
- Availability of the test environment and required equipment.
- Availability of the necessary documentation for testing.

3. Test Requirements

Test requirements include:

- The program must be able to read and parse input data files.
- The program must be able to identify and handle input parameter errors.
- The program must be able to calculate and output golf scores for each golfer.
- The program must be able to identify and handle input data errors, such as non-numeric data or invalid par values.
- The program must be able to identify and handle duplicate records for the same golf course.
- The program must be able to generate output reports and prompt the user if files already exist.
- The program must be able to handle different input and output file paths.
- The program must be able to complete processing within one minute.

4. Test Tools

The following test tools will be utilized for the testing activities:

- Java programming language
- JUnit testing framework for automated unit testing
- Git version control system for managing the source code
- Jenkins continuous integration server for automated build and testing
- Code coverage analysis tool, such as Cobertura or JaCoCo, to measure test coverage
- Bug tracking software, such as Jira or Bugzilla, for issue tracking and resolution
- Test case management tool, such as TestRail or Zephyr, for organizing and executing test cases
- Performance testing tool, such as Apache JMeter or Gatling, for load and stress testing
- Test data generation tool, such as Databene Benerator or Mockaroo, for creating realistic test data.

- The schedule and resource requirements for each tool will be identified and included in the relevant sections of the Test Plan.

5. Resource Requirements

The following resources are required:

- Test Engineers: Fout test engineers will be required to conduct the testing activities. They will be responsible for writing test cases, executing tests, logging defects, and reporting on the testing progress.
- Test Tools: The required test tools, as identified in Section 4.0, are Java, JUnit. These tools will be used to automate the testing process and improve test efficiency.
- Test Environment: The test environment will require a computer with the necessary hardware and software specifications to run the GolfScore program.
- Training: The test engineers will require training on the GolfScore program, the test tools, and the testing methodology to be employed. This training will be provided by the development team and will be conducted before the start of the testing activities.
- Documentation: The testing activities will require the use of various documents, including the Software Requirements Specification (SRS), the Product Definition Document, the Software Development Plan, and the Test Plan. These documents will be used to guide the testing process and ensure that all requirements are met.

For more detailed estimates, please see Appendix A.

6. Test Schedule

Based on the resource requirements identified in Section 5, a test schedule has been planned with a duration of 63 days. The test schedule is designed to be compatible with the project overall schedule and will be reviewed and updated as necessary throughout the testing process. Coordination with the Project Manager and Development Lead Engineers is essential in planning a realistic and workable schedule. The detailed test schedule can be found in Appendix B.

7. Risks/Mitigation

Some potential risks could include:

- Software bugs: There is a risk of software bugs that could cause the program to crash or produce incorrect results. To mitigate this risk, automated tests will be developed and executed to catch bugs early in the development process.
- Hardware failure: If the hardware being used for testing fails, it could cause delays and impact the testing schedule. To mitigate this risk, a backup hardware will be available to use and all tests and test data will be regularly backed up.
- Time constraints: The time required to execute all the tests could exceed the allotted time for testing, causing delays and impacting the project schedule. To mitigate this risk, a prioritization of tests will be done, and the most important ones will be executed first. Also, automated testing will help to reduce the testing time.

- Environment issues: Issues related to the testing environment, such as network issues, can cause delays and impact the testing schedule. To mitigate this risk, a separate testing environment will be created and used exclusively for testing.
- User error: If the user enters incorrect data or makes a mistake while using the program, it could produce unexpected results or cause the program to crash. To mitigate this risk, the program will include clear and user-friendly instructions, and automated tests will be developed to validate user inputs.

8. Metrics

The following metrics data will be collected. Some will be collected prior to, and some after product shipment.

Prior to shipment:

Effort expended during DVT, SVT and Regression

# of defects uncovered during DVT, SVT and Regression, and development phase each defect is attributable to

Test tracking S-Curve

PTR S-Curve

After shipment:

# of defects uncovered and development phase each defect is attributable to

Size of software

9. Document History

Rev. P1 - 02/21/2023 Initial Draft

10. Definitions and Acronyms

SRS: Software Requirements Specification, a document that describes the functional and non-functional requirements of the software

DVT: Development Verification Test, a type of testing that focuses on verifying the software's behavior during development

SVT: System Verification Test, a type of testing that focuses on verifying the software's behavior as a complete system

Beta: a testing phase in which the software is released to a limited group of users to gather feedback and identify any remaining issues before a wider release

Unit testing: a type of testing that focuses on verifying individual components or modules of the software in isolation

Functional testing: a type of testing that focuses on verifying that the software performs its intended functions correctly

Limits testing: a type of testing that focuses on verifying that the software can handle large or unusual inputs without crashing or producing incorrect results

Compatibility testing: a type of testing that focuses on verifying that the software works correctly with different platforms, operating systems, or other software

Performance testing: a type of testing that focuses on verifying that the software can handle a specified amount of data or requests within a certain time frame

Documentation testing: a type of testing that focuses on verifying the accuracy, completeness, and understandability of the software's documentation

PTR: Problem Tracking Report, a document used to track issues and defects found during testing

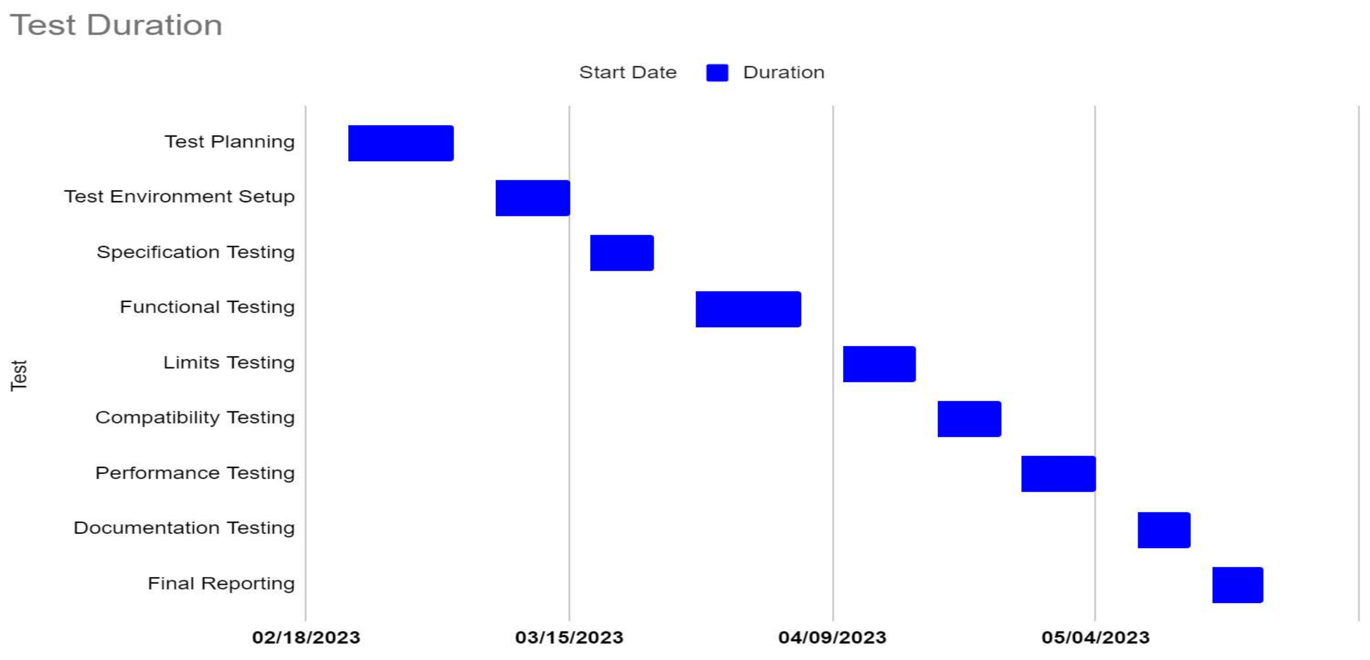
Gantt chart: a type of chart used to represent a project schedule, with activities shown as bars on a timeline

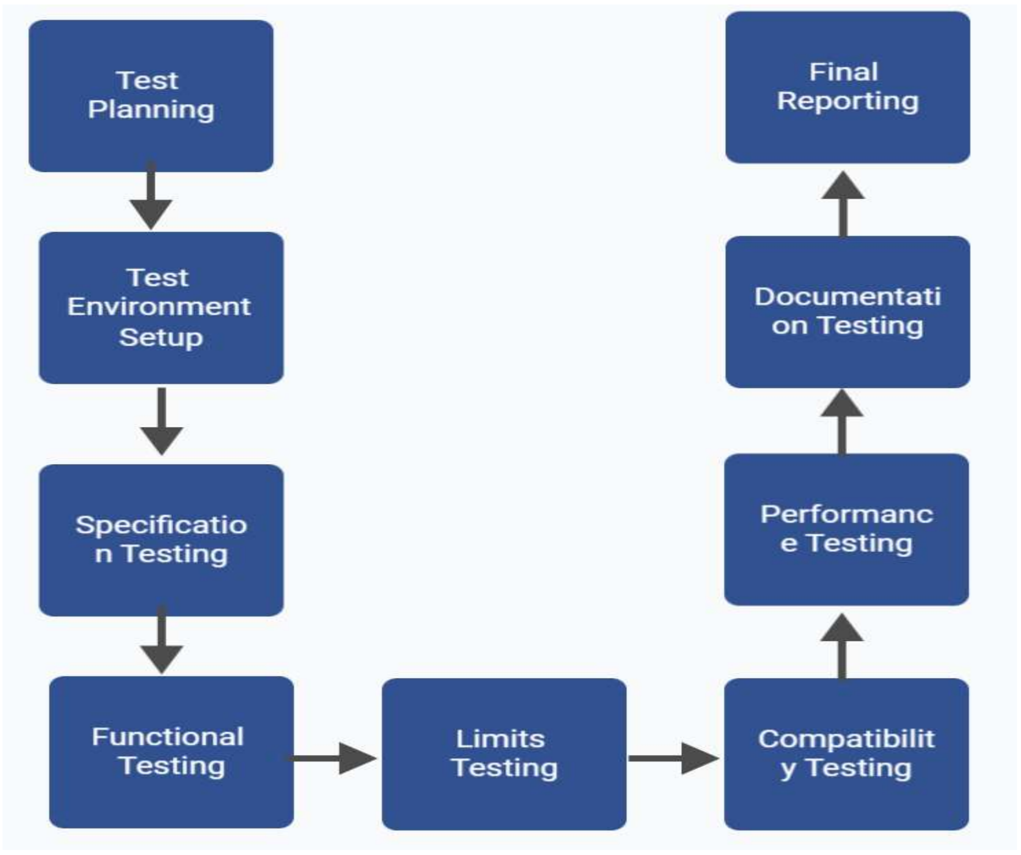
PERT chart: a type of chart used to represent a project schedule, with activities shown as nodes connected by arrows to represent dependencies between activities

Appendix A – Detailed Resource Requirements

Test Group	**Test Activity Responsible**	**Test Engineer**	**Estimated Hours**
Specification testing	Review SRS document	Roman Mokrynia	5
	Review SRS acceptance criteria	Yarik Kravchenko	4
	Create test cases based on SRS	Valentyn Sladkovenko	20
Functional testing	Input parameter testing	Volodymyr Krasnevskyy	12
	Error handling testing	Roman Mokrynia	8
	Data processing testing	Valentyn Sladkovenko	15
	Output generation testing	Yarik Kravchenko	10
Limits testing	Test handling of large data files	Volodymyr Krasnevskyy	15
	Test report generation for a large number of golfers	Roman Mokrynia	10
Compatibility testing	Test on Windows OS	Yarik Kravchenko	8
	Test on MacOS	Valentyn Sladkovenko	8
	Test on Linux	Volodymyr Krasnevskyy	8
Performance testing	Test data processing speed	Volodymyr Krasnevskyy	20
	Test report generation speed	Valentyn Sladkovenko	20
Documentation testing	Review program documentation	Yarik Kravchenko	6
	Ensure documentation accuracy and completeness	Roman Mokrynia	10
	Verify documentation is understandable	Volodymyr Krasnevskyy	8
Total Effort			**187**

Appendix B – Detailed Test Schedule





Appendix C – Test Cases

Nr.	Test Case	Test Type
1.	Verify that the program generates a Course Report in the correct format and with the correct information when given the "-c" option.	Functional
2.	Verify that the program generates a Tournament Ranking Report in the correct format and with the correct information when given the "-t" option.	Functional
3.	Verify that the program generates a Golfer Report in the correct format and with the correct information when given the "-g" option.	Functional
4.	Verify that the program can generate multiple reports at once when given multiple options.	Functional
5.	Verify that the program can handle input files with Course Records in a different order than listed in the file.	Functional
6.	Verify that the program can handle input files with Golfer Records in a different order than listed in the file.	Functional
7.	Verify that the program can handle input files with Course Records and Golfer Records intermixed.	Functional
8.	Verify that the program can handle input files with Golfer Records for courses that were not listed in the Course Records.	Functional
9.	Verify that the program can handle input files with fewer than 18 hole stroke counts for a given Golfer Record.	Functional
10.	Verify that the program can handle input files with more than 18 hole stroke counts for a given Golfer Record.	Functional
11.	Verify that the program can handle input files with non-numeric data in stroke count fields.	Functional
12.	Verify that the program can handle input files with non-numeric data in par fields.	Functional
13.	Verify that the program can handle input files with par values that are not 3, 4, or 5.	Functional
14.	Verify that the program can handle input files with duplicate Golfer Records for a given course, and that it only uses the first record.	Functional
15.	Verify that the program can handle input files with no Golfer Records for a given course.	Functional

16.	Verify that the program can handle input files with more than 5 golf courses specified for a tournament.	Functional
17.	Verify that the program can handle input files with fewer than 2 golfers entered in the tournament.	Functional
18.	Verify that the program can handle input files with more than 12 golfers entered in the tournament.	Functional
19.	Verify that the program can handle input files with negative stroke counts for a given hole.	Functional
20.	Verify that the program can handle input files with more than 3 output options.	Functional
21.	Verify that the program is able to run on the minimum system requirements specified by the developer.	Non-functional
22.	Verify that the program is able to run on different operating systems and platforms.	Non-functional
23.	Verify that the program runs quickly and does not take too long to generate reports.	Non-functional
24.	Verify that the program does not use too much memory and does not crash due to memory issues.	Non-functional
25.	Verify that the program is user-friendly and easy to use.	Non-functional
26.	Verify that the program provides helpful error messages when input or output errors occur.	Non-functional
27.	Verify that the program handles errors gracefully and does not crash when unexpected input is provided.	Non-functional
28.	Verify that the program is able to handle large input files without slowing down or crashing.	Non-functional
29.	Verify that the program is able to handle a large number of output files without slowing down or crashing.	Non-functional
30.	Verify that the program is able to write output files to different file formats.	Non-functional
31.	Verify that the program is able to handle different character encodings in input files.	Non-functional
32.	Verify that the program is able to handle different date and time formats in input and output files.	Non-functional
33.	Verify that the program is able to handle special characters and symbols in input and output files.	Non-functional
34.	Verify that the program is able to handle input files with blank or missing fields.	Non-functional

35.	Verify that the program is able to handle input files with different line endings.	Non-functional