THIS COMPANY INC.

Test Plan for GolfServ

Version 1.1

Jesus Luigi Sison

December 12, 2024

Contents

1.0 Introduction

1.1. Objective

The Test Plan provides a comprehensive approach to testing the GolfScore software (Release 1.1), ensuring the application meets all specified requirements. This document covers the entire testing effort, including unit testing, development testing, system verification, and potential beta testing.

1.2. Project Description

The Golf Scoring Software is designed to provide golfers and tournament organizers with an intuitive and efficient platform for recording and managing scores. It aims to streamline scoring processes, enhance accuracy, and provide real-time updates and analytics during golf tournaments.

1.3. Process Tailoring

This project will utilize Agile software development and management processes as a guideline. Tailoring includes:

- Excluding destructive testing, as the software is non-critical for safety.
- Prioritizing functional, performance, and compatibility testing to ensure a seamless user experience.

Testing will follow a comprehensive software quality assurance approach with the following test types:

- Test Development
- Entrance Testing
- Main Testing
- Exit Testing
- Regression Testing
- Beta Testing

1.4. Referenced Documents

- GolfScore Software Requirements Specification (SRS), Revision 1.1, dated July 18, 2017

2.0 Assumptions/Dependencies

2.1 Functional Assumptions

- Input files will comply with specified record formats
- Tournament will involve 2-12 golfers
- Tournaments will use 1-5 golf courses
- Each course will have 18 holes with par values of 3, 4, or 5

2.2 Technical Dependencies

- Development of executable must be completed prior to testing
- Test environment requires Windows 2000 or later
- Command-line interface must be fully functional
- Input/output file handling must be robust

2.3 External Dependencies

- Timely delivery of source code
- Availability of test input data sets
- Access to development and testing environments

3.0Test Requirements

3.1 Functional Requirements Testing

3.1.1. Verify correct processing of Course Records

- Validate course name parsing
- Verify course identifier assignment
- Check par value validation (3, 4, or 5)

3.1.2 Validate Golfer Records Processing

- Name parsing
- Course identifier matching
- Stroke count recording

3.1.3 Scoring Calculation Verification

- Validate scoring logic per SRS section 2.3.2
- Confirm total tournament score calculations
- Verify course-level and tournament-level scoring

3.2 Command-Line Interface Testing

3.2.1 Option Handling

- Test -h (help) display
- Validate -c, -t, -g report generation options

- Check combined option processing

3.2.2 Input Parameter Validation

- Test filename existence checks
- Verify output directory handling
- Validate error messages for invalid inputs

3.3 Report Generation Testing

3.3.1 Tournament Ranking Report

- Verify descending score order
- Check alphabetical sorting for tied scores
- Validate final standing calculations

3.3.2 Golfer Report

- Confirm alphabetical (by last name) sorting
- Validate report content matches Tournament Ranking Report

3.3.3. Course Report

- Verify per-course golfer listing
- Check hole-by-hole stroke count display
- Validate score-based descending order

3.4 Error Handling Testing

3.4.1 Input Data Validation

- Test non-numeric data detection
- Verify par value range enforcement
- Check duplicate course record handling

3.4.2 Output File Handling

- Test file overwrite prompts
- Verify new file creation
- Check error scenarios during file writing

4.0 Test Tools

Software/Tool Name	Use/Purpose
Selenium	automated functional testing
Jmeter	performance testing
BrowserStack	cross-browser and device compatibility testing
Postman	API testing
Jira	defect tracking and test case management

5.0 Resource Requirements

Resource Name	Details
Test Engineers	7
Tools	Selenium, JMeter, BrowserStack, Postman, Jira
Effort	323 man-hours
Development Environment	Cloud-based testing infrastructure
Hardware	High-performance laptops and mobile devices for compatibility testing

6.0 Test Schedule

The test schedule will begin on December 13, 2024, and include the following milestones:

1. Test Development: 10 working days
2. Entrance Testing: 10 working days
3. Main Testing: 10 working days
4. Exit Testing: 10 working days
5. Regression Testing: 10 working days
6. Beta Testing: 10 working days

7.0 Risks/Mitigation

Risk	Mitigation
Delayed code delivery	Close coordination with developers and early flagging of potential delays.
Insufficient test data	Generate demo data programmatically or thru research and validate against real-world scenarios

Tool incompatibility	Perform preliminary tool setup and testing prior to the schedule

8.0Metrics

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 testing.
- Number of defects uncovered during testing phases.
- Test tracking S-Curve.

After Shipment:

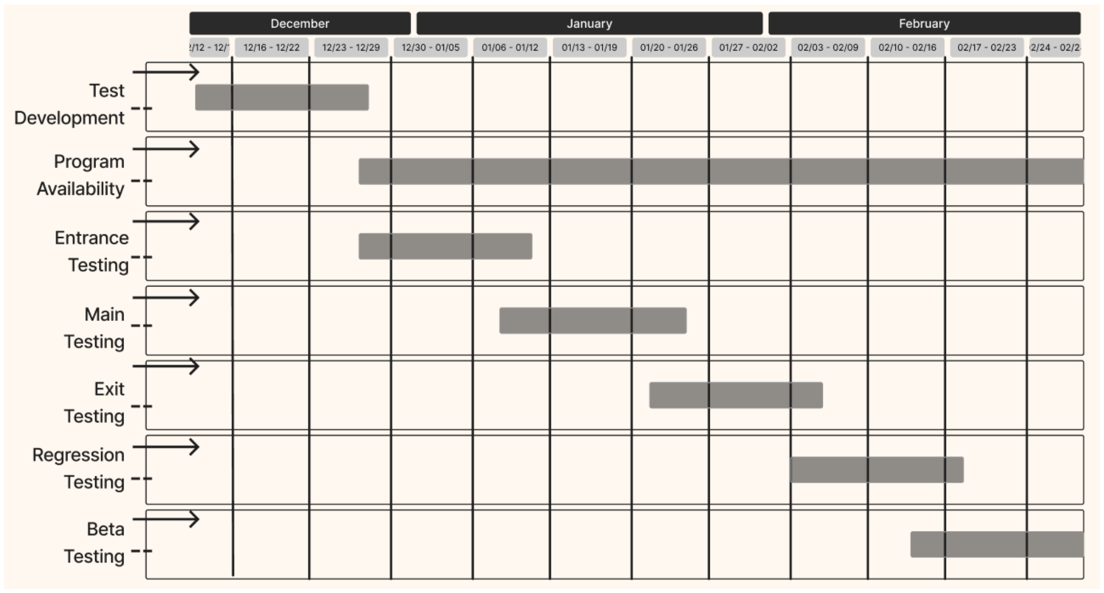
- Number of defects uncovered post-release.
- Size and complexity of software modules.

Appendix A – Detailed Resource Requirements

No.	Test Activity	No. of Personnel	Test Engineer Names	No. of Hours	Responsibilities
1	Test Development	3	• Abby • Haywood • Mark	64	Test strategy formulation, test case design, test environment setup
2	Entrance Testing	3	• Devon • Abby • Sloan	53	Initial system validation, preliminary functionality checks
3	Main Testing	5	• Devon • James • Haywood • Sloan • Abby	67	Comprehensive functional and system testing
4	Exit Testing	4	• Trey • James • Sloan • Abby	45	Final validation, acceptance criteria verification
5	Regression Testing	4	• Trey • James • Sloan • Abby	64	Ensuring no new defects introduced, backward compatibility
6	Beta Testing	4	• Trey • James • Sloan • Abby	30	Final testing before official release of a product
Total				323	

Appendix B – Detailed Test Schedule

A. Gantt Chart



B. PERT Chart

- **Activity A**: Test Development
  - Duration: 14 days (12/13 - 12/27)
  - No dependencies
- **Activity B**: Program Availability
  - Duration: 64 days (12/27 - 02/29)
  - Depends on Activity A (Program Initialization)
- **Activity C**: Entrance Testing
  - Duration: 14 days (12/27 - 1/10)
  - Depends on Activity B (Entrance Test Data)
- **Activity D**: Main Testing
  - Duration: 14 days (01/11- 01/25)
  - Depends on Activity C (Passed Data from the Entrance Testing)
- **Activity E**: Exit Testing
  - Duration: 14 days (01/26 - 02/09)
  - Depends on Activity D (Complete Data)
- **Activity F**: Regression Testing
  - Duration: 14 days (02/09 - 02/23)
  - Depends on Activity E
- **Activity G**: Beta Testing
  - Duration: 12 days (02/17 - 02/28)
  - Depends on Activity F

Appendix C – Test Cases

No.	Test Case	Test Type	Expected Result
1	The program shall be Written in C or C++.	Non-Functional	Source code review verifies the program is implemented in C or C++
2	The program shall run on a PC running Windows 2000	Non-Functional	Program executes without errors on a PC with Windows 2000
3	The program shall run on PC running Windows XP	Non-Functional	Program executes without errors on a PC with Windows XP
4	The program shall run on PC running Windows VISTA	Non-Functional	Program executes without errors on a PC with Windows VISTA
5	The program shall run on PC running Windows 7	Non-Functional	Program executes without errors on a PC with Windows 7
6	The program shall run on PC running Windows 8	Non-Functional	Program executes without errors on a PC with Windows 8
7	The program shall run on PC running Windows 10	Non-Functional	Program executes without errors on a PC with Windows 10
8	The program shall run on PC running Windows 11	Non-Functional	Program executes without errors on a PC with Windows 11
9	The program shall run as a stand-alone executable	Non-Functional	Program can execute independently without requiring external dependencies or installations.
10	Command line options -ctg shall be accepted	Functional	Program accepts –ctg as a valid input parameter without errors.
11	Command line options –c shall be accepted	Functional	Program accepts -c as a valid input parameter without errors.
12	Command line options –t shall be accepted	Functional	Program accepts -t as a valid input parameter without errors.
13	Command line options –g shall be accepted	Functional	Program accepts -g as a valid input parameter without errors.
14	The number of golf course 1 shall be accepted	Functional	Program accepts input 1 as a valid number of golfers
15	The number of golf course 5 shall be accepted	Functional	Program accepts input 5 as a valid number of golfers
16	The number of golf course -5 shall be accepted	Functional	Program accepts input -5 as a valid number of golfers
17	The number of golf course 0 shall return an error	Functional	Program rejects input 0 and displays an error message.
18	The number of golfers 1 shall return an error	Functional	Program rejects input 1 and displays an error message.
19	The number of golfers 2 shall be accepted	Functional	Program accepts input 2 as a valid number of golfers
20	The number of golfers 12 shall be accepted	Functional	Program accepts input 12 as a valid number of golfers
21	The number of golfers 13 shall return an error	Functional	Program rejects input 13 and displays an error message.
22	Par for hole 2 shall return error	Functional	Program rejects input 2 for hole par and displays an error message.
23	Par for hole 6 shall return error	Functional	Program rejects input 6 for hole par and displays an error message.

24	Par for hole 3 shall be accepted	Functional	Program accepts input 3 as a valid par value for the hole.
25	Par for hole 4 shall be accepted	Functional	Program accepts input 4 as a valid par value for the hole.
26	Par for hole 5 shall be accepted	Functional	Program accepts input 5 as a valid par value for the hole.
27	Calling the program with command line option –ctg shall generate 3 output file trank.rep, golfer.rep, course.rep. If any	Functional	Program generates the specified output files if they do not exist.
29	Calling the program with command line option –c shall generate output file course.rep. If file already exit the user shall prompted with a message that say file already exits and asking it to overwrite it or not.	Functional	Generates specific output file. If the file exists, the user is prompted to overwrite or not.
30	Calling the program with command line option –t shall generate output file trank.rep. If file already exit the user shall prompted with a message that say file already exits and asking it to overwrite it or not.	Functional	Generates specific output file. If the file exists, the user is prompted to overwrite or not.
31	Calling the program with command line option –g shall generate output file golfer.rep. If file already exit the user shall prompted with a message that say file already exits and asking it to overwrite it or not.	Functional	Generates specific output file. If the file exists, the user is prompted to overwrite or not.
32	If output cannot be saved due to insufficient permission the program shall display error.	Functional	Program displays an error message indicating insufficient permissions.