**MSSE SOFTWARE, INC.**

**Test Plan for**

**GolfScore Revision 1.1**

**31/05/2022**

**Yan Guchek**

**Test Group**



Contents

1.0 Introduction **2**

1.1. Objective **2**

1.2. Project Description **3**

1.3. Process Tailoring **[3](#_2et92p0)**

1.4. Referenced Documents **6**

2.0 Assumptions/Dependencies **6**

3.0 Test Requirements **7**

4.0 Test Tools **7**

5.0 Resource Requirements **8**

6.0 Test Schedule **8**

7.0 Risks/Mitigation **8**

8.0 Metrics **9**

9.0 Definitions and Acronyms **9**

Appendix A – Detailed Resource Requirements **10**

Appendix B – Detailed Test Schedule **11**

Appendix C – Test Cases **17**

# Introduction

## Objective

This document describes the test plan for the GolfScore Revision 1.1 and includes information on what is to be tested, and how the testing is to be performed and what is not to be tested. Specifically, this document describes the tests to be performed, the testing schedule, resources required, dependencies, test tools and metrics. Changes in requirements and the structure of a team must always reflect in this document.

The main focus of this test is to verify the software requirements specification (SRS) for the GolfScore program as contained in Appendix A.

## Project Description

GolfScore is a program used to generate golf tournament results for golfers along each course. This program takes an input text file (as described in the SRS) and produces three output text files (also described in the SRS).

## Process Tailoring

## 1.3.1. Description

The V-model will be used in the work on this project. The company chose it for a number of reasons (works well for small projects; testing starts from the very beginning, so ambiguities are identified from the very beginning; it is easy to manage as each stage has clearly defined goals and objectives). In accordance with this, our test team, together with the developers, plans to start testing from the very beginning of development.

To reduce the time for regression testing, we will gradually automate tests that cover the stable parts of the application.

The following tests are planned for the project:

* specification and documentation review;
* smoke testing (including: the correctness of launch, the correctness of closing the program, displaying the logo and version while the program is running);
* unit, integration and system;
* white and black box;
* functional (including boundaries, equivalence classes, field validation testing),
* compatibility,
* performance,
* beta/acceptance.

## 1.3.1.1. Features not to be tested

What we do not plan to test: opening and entering data into an input .txt file and reading and opening output .rep files, since this happens in a third-party application.

Visual testing is not carried out as there is no GUI.

There will be no stress testing either. The requirements do not include the ability to process more than one tournament at a time.

There is no security testing either. All incoming data is in another application file.

## 1.3.1.2. Documentation review

First of all, a review of all available project documentation will be carried out. During the testing process, logical errors in the formulation of the tasks, discrepancies in the requirements can be identified, and a checklist and a list of checks on the provided requirements can be drawn up.

Various specialists will be involved in the documentation testing process: testers, project managers, business analysts, developers.

**Example of checks**

1. Calling Golfscore: What if we enter not the name of the folder, but the full path to the output?

2. Scoring: For the hole with

3 strokes par 6 points cannot be obtained according to the requirements, as it is necessary to make 0 hits. Requirements should be clarified.

3. Input: The requirements must specify the exact format of the golfers' names. This is required to achieve several goals: to prevent possible duplicates, to correctly sort by last name in the output file, and to prevent going beyond 19 characters.

4. Input: The requirements must specify the exact format of the courses' names. This is to prevent going beyond 18 characters.

5. Input: The requirements must specify the exact format of the courses' IDs.

## 1.3.1.3. Unit testing

There will be four units to be tested: Calling GolfScore, Scoring, Data Input Recognition, and Data Output. Our 2 ATs, together with the developers, will work on unit test cases, but first of all, they will create an opportunity for manual testers to refer to each unit using a command line prompt.

**Test cases examples:**

| **Task title** | **Input data** | **Expected result** |
| --- | --- | --- |
| Positive testing of Calling Golfscore | -tcg | Generate the Course Report, the Tournament Ranking report, and Golfer Report. |
| Negative testing of Calling Golfscore | [input file and/or output directory doesn’t exist] | Input parameter error be reported. |
| Testing of Scoring (equivalence classes and borders for stroke count for hole with 3 strokes par: >3; 3; 2; 1; 0) | 1 | 4 points |

## 1.3.1.4. Integration testing

Integration testing can be carried out when all test cases for all units have been successfully passed.

The Integration testing will be broken into three phases:

* The correctness of the joint functioning of the key elements of the application (Data Input and Scoring) will be checked first. The data entered into the in.txt file will be tested against the data used for scoring.
* The consistency of the Scoring data and the Data output will be tested.
* Testing the correctness of calculations in the output file with data from the input file.
* The correctness of the joint functioning of the GolfScore, Data Input, and Data Output will be tested.

**Test cases examples:**

| **Task title** | **Input data** | | **Expected result** | |
| --- | --- | --- | --- | --- |
| Candidate for automation Testing of Data Output (Tournament Report: sorting of golfers by score and alphabet) | Wira Osgar  Vayu Nora  Abel Domitila  Aileen Hamo  Maarika Babylas  Ea Sixtine  Roddy Trude  Amadeo Nil | 18  13  7  21  13  16  14  16 | Aileen Hamo  Wira Osgar  Amadeo Nil  Ea Sixtine  Roddy Trude  Maarika Babylas  Vayu Nora  Abel Domitila | 21  18  16  16  14  13  13  7 |
| Testing the correctness of calculations in the output file with data from the input file (equivalence classes and borders for Courses number: <1; 1-5; >5) | 3 | | [the entered data is correct, no issues] | |
| Testing the correctness of calculations in the output file with data from the input file (equivalence classes and borders for stroke count for hole with 5 strokes par: >5; 5; 4; 3; ≤2) | 1 | | 6 points | |

## 1.3.1.5. System testing

The system testing will be divided into two phases:

* Main Test. At this stage, not only functional testing will be carried out, but also an assessment of the quality characteristics of the system - its stability, reliability and performance. Load testing: the ability to process everything within a minute (SRS, 4). Performance: The ability to work during a tournament lasting as long as the longest known tournament.
* The Regression Test will run as a subset of the main test to validate all the functionality of the application after all issues found during the main testing have been fixed. This means that all severity 1 and 2 defects have been fixed and verified, and that the product is ready for release.\

**Test cases examples**

| **Task title** | **Input data** | **Expected result** |
| --- | --- | --- |
| Performance testing | [The application was launched and left running for the duration of the longest known golf course] | The application continues its work, responds to requests and generates reports without having to restart it |
| Load testing | [All three types of report files are requested from the application] | The application performed all calculations and the files were generated within one minute |
| Compatibility testing | [The application is launched on Windows 7] | The application is running. The screen displays the name and version of the application. |

## 1.3.1.6. Acceptance Testing

The final stage will be beta testing by the crowd testers of the UTest platform selected according to the appropriate criteria. Selection criteria for the Special Requirements Survey: experience in viewing golf tournaments and analyzing the standings. Testing will consist of executing test cases and writing a review.

Review questions examples

1. Have you successfully used all the options of the program (help, output of results)? Describe in detail all difficulties, if any.

2. Which feature(s) did you like the most? Which feature(s) would you like to see? Please write at least 1 feature you liked the most and 1 feature you would like to see (if you have).

3. How clear do the provided counting results seem to you?

4. How do you think the Tournament Ranking Report and the Golfer Report differ? Briefly describe the key differences.

5. Imagine that the results of the program output are displayed on the screen during the broadcast of the tournament. How should they look and where should they be located? At what point should this or that data appear? Describe your wishes in detail.

## Referenced Documents

SRS: Bitzengolfer N. Software Requirements Specification/Design Document for the GolfScore. Revision 1.1, 7/18/2017.

# Assumptions/Dependencies

As described above, our test team will fully review the project, according to V-model testing, but we assumed that the development team unit test their code while developing the software.

For confirmation with the set schedule, the program must be made available by the development team by September 9, 2022.

# Test Requirements

Entrance Tests:

* The program is written in either C or C++.
* The program runs on a PC running Windows 2000 or any later version.
* The program will run as a stand-alone executable.
* The program can be run from the command line prompt.
* The program is run with valid input parameters

Main Tests:

* The number of golf courses specified for the tournament must be from 1 to 5.
* Each golfer is expected to play each course once.
* The number of golfers entered in the tournament can be from 2 to 12.
* Par for holes on each course must be either 3, 4, or 5.
* Score earned by a golfer for each hole played is between 0 and 6 (0 and 6 included).
* The first set of records in the input file (course records) exist and follow the specified format for each entry.
* There is a delimiter record that signals the end of course records.
* A second set of records (golfer records) exist in the input file and each entry follows the specified format.
* There is a delimiter record that signals the end of the input file.

Exit Tests:

* The program should produce a number of reports corresponding to the specified options.
* The generated reports should be saved as text files in the specified output directory (or if not specified, in the directory of the input file) with the extension “.rep”.
* If requested, the tournament ranking report should contain a list of all golfers in the specified format. The list should be in descending order of final score and should be saved with an output filename of trank.rep.
* If requested, the golfer report should contain a list of all golfers in the specified format. The list should be alphabetical with respect to the golfers’ last name and should be saved with an output filename of golfer.rep.
* If requested, the course report should contain a section for each Golf Course listed in the input Course Records in the specified format. It should be saved with

# Test Tools

To aid the testing process, the following testing tools are required:

* Defect reporting and tracking software
* Installation media for multiple Windows versions above 2000 (e.g. XP, Vista, 7, 8, 8.1, & 10)

# Resource Requirements

Also, our testing department currently employs three junior manual testers (Ya Kondwani = MT-1, Jescha Annemieke = MT-2, Milorad Jacopo = MT-3), two middle automation testers (Allan Theo = AT-1,Marcia Darija = AT-2) and the test team lead (Yan Guchek = TTL).

It is planned to carry out the work without involving outside human resources. The only exception is beta testing (see above: 1.3.1.6. Acceptance Testing).

Pay attention at Appendix A for details.

Unit tests stage: 1994 working hours = two and a half months of work of our team of five people on the condition of an 8-hour working day and a five-day working week.

Integration tests stage: 790 working hours = one month.

System tests stage: 276 working hours = third of the month.

Acceptance tests stage: 60 working hours = a day and a half.

**A grand total of the effort: 3120 hours = four months.**

# Test Schedule

The testing defined in this document shall be completed according to the following schedule. See Appendix A for details.

| Test Sequence | Start | Finish |
| --- | --- | --- |
| Documentation review | 06/01/2022 | 06/05/2022 |
| Additional discussion of support for Windows 11 | 06/08/2022 | 06/08/2022 |
| Mitigation plan development | 06/09/2022 | 06/06/2022 |
| Unit testing | 06/17/2021 | 08/31/2022 |
| Integration testing | 09/01/2022 | 10/02/2022 |
| System testing | 10/03/2022 | 10/14/2022 |
| Acceptance testing | 10/15/2022 | 10/17/2022 |

# Risks/Mitigation

* Without having a program that enforces compliance in the structure of input data, there’s a high probability of input data errors.
* Assembling the right team for the Acceptance testing can take longer than a month. If there are at least 5 people in the team by the specified deadline, it is worth not shifting the timing of searching for new members, but distributing all test cases among the existing participants and extending the deadline by 2 days.
* Changes to the original requirements or designs.

# 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

# 9.0 Definitions and Acronyms

| SRS | Software Requirements Specification |
| --- | --- |
| IDE | Integrated Development Environment |
| MT | Manual Tester |
| AT | Automation Tester |
| TTL | Test Team Lead |

**Appendix A – Detailed Resource Requirements**

* PCs that are capable of hosting virtual machines are required such that the program can be tested on multiple versions of Windows.
* Virtualization software is required such that multiple versions of Windows can be installed to test the program.

**Appendix B – Detailed Test Schedule**

| **Task ID** | **Task Title** | **TTL** | **MT-1** | **MT-2** | **MT-3** | **AT-1** | **AT-2** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| U1 | Documentation review | 136 | 16 | 16 | 16 | 16 | 16 |
| U2 | Additional discussion of support for Windows 11 | 8 |  |  |  | 4 | 4 |
| U3 | Mitigation plan development | 48 |  |  |  |  |  |
| U4 | Developing unit-tests | 80 |  |  |  | 80 | 42 |
| U5 | Developing the interface for manual testers for Calling Golfscore | 1 |  |  |  | 60 | 60 |
| U6 | Developing the interface for manual testers for Scoring | 1 |  |  |  | 60 | 60 |
| U7 | Developing the interface for manual testers for Input Data Recognition | 1 |  |  |  | 60 | 60 |
| U8 | Developing the interface for manual testers for Output Data | 1 |  |  |  | 60 | 60 |
| U9 | Smoke testing of Calling Golfscore |  | 4 | 4 | 4 | 2 |  |
| U10 | Positive testing of Calling Golfscore (14 combinations of options) |  | 8 | 8 | 8 | 2 |  |
| U11 | Positive testing of Calling Golfscore (input file) |  | 8 | 8 | 8 | 2 |  |
| U12 | Positive testing of Calling Golfscore (output directory) |  | 8 | 8 | 8 | 2 |  |
| U13 | Positive testing of Calling Golfscore (“-h” + input filename and/or output directory) |  | 8 | 8 | 8 | 2 |  |
| U14 | Negative testing of Calling Golfscore (digits, commas, spases, non-latin and other non-valid characters) |  | 8 | 8 | 8 | 2 |  |
| U15 | Negative testing of Calling Golfscore (input file and/or output directory doesn’t exist) |  | 8 | 8 | 8 | 2 |  |
| U16 | Smoke testing of Scoring |  | 4 | 4 | 4 |  | 2 |
| U17 | Testing of Scoring (equivalence classes and borders for Courses number: <1; 1-5; >5) |  | 8 | 8 | 8 |  | 2 |
| U18 | Testing of Scoring (equivalence classes and borders for Golfers number: <2; 2-12; >12) |  | 8 | 8 | 8 |  | 2 |
| U19 | Testing of Scoring (equivalence classes and borders for Holes number per course: <18; 18; >18) |  | 8 | 8 | 8 |  | 2 |
| U20 | Testing of Scoring (equivalence classes and borders for par per hole: <3; 3 -5; >5) |  | 8 | 8 | 8 |  | 2 |
| U21 | Testing of Scoring (equivalence classes and borders for stroke count for hole with 3 strokes par: >3; 3; 2; 1; 0) |  | 8 | 8 | 8 |  | 2 |
| U22 | Testing of Scoring (equivalence classes and borders for stroke count for hole with 4 strokes par: >4; 4; 3; 2; 1) |  | 8 | 8 | 8 |  | 2 |
| U23 | Testing of Scoring (equivalence classes and borders for stroke count for hole with 5 strokes par: >5; 5; 4; 3; ≤2) |  | 8 | 8 | 8 |  | 2 |
| U24 | *Candidate for automation*  Testing of Scoring (golfer score for course) |  | 8 | 8 | 8 |  | 16 |
| U25 | *Candidate for automation*  Testing of Scoring (stroke count for  course) |  | 8 | 8 | 8 |  | 16 |
| U26 | *Candidate for automation*  Testing of Scoring (golfer total tournament  score) |  | 8 | 8 | 8 |  | 16 |
| U27 | Smoke testing of Input Data Recognition |  | 4 | 4 | 4 | 2 |  |
| U28 | Testing of Input Data Recognition (equivalence classes and borders for Course Records columns: ≤1; 2 -19; 20; 21 - 38; >38) |  | 8 | 8 | 8 | 2 |  |
| U29 | Positive testing of Input Data Recognition (field validation for Course Records) |  | 8 | 8 | 8 | 2 |  |
| U30 | Negative testing of Input Data Recognition (field validation for Course Records) |  | 8 | 8 | 8 | 2 |  |
| U31 | Testing of Input Data Recognition (equivalence classes and borders for Golfer Records columns: ≤1; 1; 2; 3 -9; 10 -29; 30; 31 -48; >48) |  | 8 | 8 | 8 | 2 |  |
| U32 | Positive testing of Input Data Recognition (field validation for Golfer Records) |  | 8 | 8 | 8 | 2 |  |
| U33 | Negative testing of Input Data Recognition (field validation for Golfer Records) |  | 8 | 8 | 8 | 2 |  |
| U34 | Smoke testing of Data Output |  | 4 | 4 | 4 |  | 2 |
| U35 | Testing of Data Output (overwrighting of an existing file) |  | 8 | 8 | 8 |  | 2 |
| U36 | Testing of Data Output (separate promt for each type of report) |  | 8 | 8 | 8 |  | 2 |
| U37 | Testing of Data Output (creation of non - existing file) |  | 8 | 8 | 8 |  | 2 |
| U38 | Unit testing results analysis | 20 | 8 | 8 | 8 | 4 | 2 |
| U39 | Regression testing | 48 | 48 | 48 | 48 | 48 |  |
| Int1 | Smoke testing of Calling Golfscore + Input |  | 4 | 4 | 4 | 2 | 2 |
| Int2 | Smoke testing of Calling Golfscore + Output |  | 4 | 4 | 4 |  | 16 |
| Int3 | Candidate for automation  Testing of Data Output (Tournament  Report: score for each course) |  | 8 | 8 | 8 |  | 16 |
| Int4 | Candidate for automation  Testing of Data Output (Tournament  Report: total score) |  | 8 | 8 | 8 |  | 16 |
| Int5 | Candidate for automation  Testing of Data Output (Tournament  Report: final standings) |  | 8 | 8 | 8 |  | 16 |
| Int6 | Candidate for automation  Testing of Data Output (Tournament  Report: sorting of golfers by score and  alphabet) |  | 8 | 8 | 8 |  | 16 |
| Int7 | Candidate for automation  Testing of Data Output (Golfer Report:  sorting of golfers by last name) |  | 8 | 8 | 8 |  | 16 |
| Int8 | Candidate for automation  Testing of Data Output (Course Report:  total score) |  | 8 | 8 | 8 |  | 16 |
| Int9 | Candidate for automation  Testing of Data Output (Course Report:  stroke count) |  | 8 | 8 | 8 |  | 16 |
| Int10 | Candidate for automation  Testing of Data Output (Course Report:  sorting of golfers) |  | 8 | 8 | 8 |  | 16 |
| Int11 | Testing the correctness of calculations in the output file with data from the input file (equivalence classes and borders for Courses number: <1; 1 -5; >5) |  | 8 | 8 | 8 | 2 |  |
| Int12 | Testing the correctness of calculations in the output file with data from the input file (equivalence classes and borders for Golfers number: <2; 2 -12; >12) |  | 8 | 8 | 8 | 2 |  |
| Int13 | Testing the correctness of calculations in the output file with data from the input file 8 8 8 2 14 (equivalence classes and borders for Holes number per course: <18; 18; >18) |  | 8 | 8 | 8 | 2 |  |
| Int14 | Testing the correctness of calculations in the output file with data from the input file (equivalence classes and borders for par per hole: <3; 3-5; >5) |  | 8 | 8 | 8 | 2 |  |
| Int15 | Testing the correctness of calculations in the output file with data from the input file (equivalence classes and borders for stroke count for hole with 3 strokes par: >3; 3; 2; 1; 0) |  | 8 | 8 | 8 | 2 |  |
| Int16 | Testing the correctness of calculations in the output file with data from the input file (equivalence classes and borders for stroke count for hole with 4 strokes par: >4; 4; 3; 2; 1) |  | 8 | 8 | 8 | 2 |  |
| Int17 | Testing the correctness of calculations in the output file with data from the input file (equivalence classes and borders for stroke count for hole with 5 strokes par: >5; 5; 4; 3; ≤2) |  | 8 | 8 | 8 | 2 |  |
| Int18 | Regression testing | 48 | 48 | 48 | 48 | 48 |  |
| Int19 | Integration testing results analysis | 20 |  |  |  |  |  |
| S1 | Smoke testing |  | 4 | 4 | 4 | 2 |  |
| S2 | Performance testing |  | 4 | 4 | 4 | 2 |  |
| S3 | Load testing |  | 4 | 4 | 4 | 2 |  |
| S4 | Compatibility testing (Windows 2000) |  | 4 | 4 | 4 | 2 |  |
| S5 | Compatibility testing (Windows XP) |  | 4 | 4 | 4 | 2 |  |
| S6 | Compatibility testing (Windows Vista) |  | 4 | 4 | 4 | 2 |  |
| S7 | Compatibility testing (Windows 7) |  | 4 | 4 | 4 | 2 |  |
| S8 | Compatibility testing (Windows 8) |  | 4 | 4 | 4 | 2 |  |
| S9 | Compatibility testing (Windows 8.1) |  | 4 | 4 | 4 | 2 |  |
| S10 | Compatibility testing (Windows 10) |  | 4 | 4 | 4 | 2 |  |
| S11 | Compatibility testing (Windows 11) *if required* |  | 4 | 4 | 4 | 2 |  |
| S12 | Regression testing |  | 4 | 4 | 4 | 2 |  |
| S13 | System testing results analysis | 20 | 4 | 4 | 4 | 4 | 4 |
| S14 | Artifacts producing: test summary report, and updated traceability matrices | 48 | 4 | 4 | 4 | 4 | 4 |
| Acc1 | Acceptance testing (with outsource) | 20 | 4 | 4 | 4 |  |  |
| Acc2 | Acceptance testing results analysis | 20 | 4 | 4 | 4 | 4 | 4 |
|  | **A grand total of the effort: 3120 hours** | **520** | **520** | **520** | **520** | **520** | **520** |

**Appendix C – Test Cases**

| **Test No.** | **Test Case** | | | | **Test Type** | |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | The program shall be written in C or C++ | | | | Non-functional | |
| 2 | The program shall run on a PC running Windows 2000 | | | | Non-functional | |
| 3 | The program shall run on a PC running Windows XP | | | | Non-functional | |
| 4 | The program shall run on a PC running Windows Vista | | | | Non-functional | |
| 5 | The program shall run on a PC running Windows 7 | | | | Non-functional | |
| 6 | The program shall run on a PC running Windows 8 | | | | Non-functional | |
| 7 | The program shall run on a PC running Windows 8.1 | | | | Non-functional | |
| 8 | The program shall run on a PC running Windows 10 | | | | Non-functional | |
| 9 | The program shall run as a stand-alone executable | | | | Non-functional | |
| 10 | The program shall run from the command line prompt | | | | Non-functional | |
| 11 | Command line options “-ctg” shall be accepted | | | | Functional | |
| 12 | Command line option “-c” shall be accepted | | | | Functional | |
| 13 | Command line option “-t” shall be accepted | | | | Functional | |
| 14 | Command line option “-g” shall be accepted | | | | Functional | |
| 15 | Command line options “-c –t -g” shall be accepted | | | | Functional | |
| 16 | Command line option “-k” shall display an “unrecognizable options” message | | | | Functional | |
| 17 | Command line option “-j” shall display an “unrecognizable options” message | | | | Functional | |
| 18 | Command line option “-kj” shall display an “unrecognizable options” message | | | | Functional | |
| 19 | Command line option “-ckj” shall display an “unrecognizable options” message | | | | Functional | |
| 20 | Specifying an input filename that does not exist shall display an “input parameter error” | | | | Functional | |
| 21 | Specifying an output directory that does not exist shall display an “input parameter error” | | | | Functional | |
| 22 | Command line option “-g” shall be accepted and shall display help information | | | | Functional | |
| 23 | Calling the program as “golf -ctg in.txt golfout” where “in.txt” exists and is valid and folder “golfout” exists shall be accepted | | | | Functional | |
| 24 | Calling the program as “golf -ctg in.txt golfout dis” where “in.txt” exists and is valid and folder “golfout” exists shall be accepted | | | | Functional | |
| 25 | Calling the program as “golf -ctg in.txt golfout” where “in.txt” exists and is valid and folder “golfout” does not exist shall display an “input parameter error” | | | | Functional | |
| 26 | Calling the program as “golf -ctg in.txt golfout” where “in.txt” does not exist shall display an “input parameter error” | | | | Functional | |
| 27 | The number of golf course “1” shall be accepted | | | | Functional | |
| 28 | The number of golf course “5” shall be accepted | | | | Functional | |
| 29 | The number of golf course “-5” shall return an error | | | | Functional | |
| 30 | The number of golf course “6” shall return an error | | | | Functional | |
| 31 | The number of golf course “0” shall return an error | | | | Functional | |
| 32 | Having multiple records for a golfer on the same golf courses shall be accepted, although a message should be displayed indicating this. The first record shall be used and processing shall continue | | | | Functional | |
| 33 | The number of golfers “0” shall return an error | | | | Functional | |
| 34 | The number of golfers “1” shall return an error | | | | Functional | |
| 35 | The number of golfers “2” shall be accepted | | | | Functional | |
| 36 | The number of golfers “12” shall be accepted | | | | Functional | |
| 37 | The number of golfers “13” shall return an error | | | | Functional | |
| 38 | Par for hole “2” shall return an error | | | | Functional | |
| 39 | Par for hole “6” shall return an error | | | | Functional | |
| 40 | Par for hole “3” shall be accepted | | | | Functional | |
| 41 | Par for hole “4” shall be accepted | | | | Functional | |
| 42 | Par for hole “5” shall be accepted | | | | Functional | |
| 43 | Golfer score per hole “7” shall return an error | | | | Functional | |
| 44 | Golfer score per hole “-1” shall return an error | | | | Functional | |
| 45 | Golfer score per hole “0” shall be accepted | | | | Functional | |
| 46 | Input data with non-numeric data where numeric data is expected shall return an error | | | | Functional | |
| 47 | Input data with numeric data where non-numeric data is expected shall return an error | | | | Functional | |
| 48 | Input data that violates delimiter constraints shall return an error | | | | Functional | |
| 49 | Input file that does not contain course records shall return an error | | | | Functional | |
| 50 | Input file that does not contain golfer records shall return an error | | | | Functional | |
| 51 | Calling the program with command line options “-ctg” shall generate 3 output files: “trank.rep”, “golfer.rep”, “course.rep”. If any of the files already exist, the user shall be prompted with a message that says the file already exists and asking whether to overwrite it or not | | | | Functional | |
| 52 | Calling the program with command line option “-c” shall generate an output file: “course.rep”. If the file already exists, the user shall be prompted with a message that says the file already exists and asking whether to overwrite it or not. | | | | Functional | |
| 53 | Calling the program with command line option “-t” shall generate an output file: “trank.rep”. If the file already exists, the user shall be prompted with a message that says the file already exists and asking whether to overwrite it or not. | | | | Functional | |
| 54 | Calling the program with command line option “-g” shall generate an output file: “golfer.rep”. If the file already exists, the user shall be prompted with a message that says the file already exists and asking whether to overwrite it or not. | | | | Functional | |
| 55 | If output cannot be saved due to insufficient permissions, the program shall display an error. | | | | Functional | |