Metrics: Test Plan

Zach Sharpe

Colorado Technical University

Table of Contents

Software Testing Environment 4

Testing Environment Summary: 4

Environment: 4

Personnel: 4

RTM: 5

The Importance of Unit / Usability / System Test Planning Process 7

Unit Testing 7

Usability Testing 8

System Testing 8

Characteristics of Test Plans 9

Test Planning Approach 10

Black Box Testing: 10

White Box Testing: 10

Summary: 10

Outline for the Test Plan 11

Scope (identification, system overview) 11

System description and operation (name of test sites, test configuration) 11

Test identification & test levels 11

Planned tests & test schedules 12

Requirement traceability 13

Regression Testing 17

Why Regression Testing 17

Potential Impact Requiring Regression Testing 17

How We Use Regression Testing 18

Revision Log for Unit 4 19

Summary 20

References 21

# Software Testing Environment

## Testing Environment Summary:

### Environment:

Our physical environment requirements allow for a healthy work environment over a long-term work relationship. We will be following guidelines from OSHA via their site: <http://www.osap.org/?page=guideosha>

The physical working environment contributes to efficiency and happiness as well. We are aiming to create a place that is inductive to work at the end of the day.

The testing environment must be equipped with reliable machines and a focused working area. This includes:

* Computers with sufficient specifications for system and component testing.
* Desks with sufficient space.
* Office chairs.
* Keyboard, mouse, and other required peripherals

### Personnel:

Testing personnel must have minimum competencies:

* Background and understanding of component tests to help with white box testing techniques and regressions testing.
* Experience utilizing popular testing frameworks such as JUnit for quick learning and integration to our existing development teams.
* For systematic, or stress testing, must have experience testing systems utilizing aggressive contrasting test cases. This includes performance testing for application run times, optimization, and general quality testing.
* Finally, we require security testing for integrity of the product as whole. Person must have experience testing web applications for security flaws and minor system testing experience.

## RTM:

The matrix used below was retrieved from StrongQA.com 1 and parsed down.

|  |  |  |
| --- | --- | --- |
| **Requirements Traceability Matrix** |  |  |
|  | | |
| **Project Name** | | **Metric** |
| **Project Manager** | | **Zach Sharpe** |
| **QA Lead** | | **Zach Sharpe** |
| **Requirement ID** | **Category/Functional Activity** | **Requirement Description** |
| 1 | Systematic Input | The software must be able to take in multiple fields and columns of a spreadsheet and parse the data into local data structures. |
| 2 | Display Output | It must be able to display this data back after parsing. |
| 3 | System Process | It must calculate averages and display them. Averages will not need to be performed for data averages that already exist. |
| 4 | Display Output | The data must be displayed in a visually, friendly manner. Ideally, a GUI will be used to display text to the user. |
| 5 | Systematic Output | Export to spreadsheet |
|  |  |  |

# The Importance of Unit / Usability / System Test Planning Process

Here, we will pair a little bit of the ‘why’ behind the testing we are planning on both systematically and the tests that allow us to align our expectations with our deliverables. Testing is an incredibly important last step to the development process and ensures a quality process. The outline below goes into a bit more detail.

Lastly, we’ll be utilizing agile testing practices. A main approach here will be inline testing which means no test crunch:

“Testing activities are planned alongside development activities. The team’s delivery succeeds if, and only if; the software is implemented and tested. The mere presence of a tester tends to result in the team asking: “How do we test this?” which in turn leads to testable software.” 2

## Unit Testing

Unit testing is crucial to the quality of our code. This type of testing allows us to take a look at the actual body of code and its functionality. This will be the most *literal* form of testing, as well as the one that ensures a working piece of software.

Unit testing can contribute to several factors, not just for release, but also for ongoing maintenance. This assists with *support testing* or more hands on testing in some of the following ways:

* Does the product run efficiently?
* We’ll perform extreme testing for fringe cases such as disk capacity, fast button clicks, and overloading some functions.
* How maintainable are the modules included

## Usability Testing

This type of testing is arguably the most important as this considers how the application interfaces and interacts with the user. The US government broadly describes this as “Usability testing refers to evaluating a product or service by testing it with representative users. Typically, during a test, participants will try to complete typical tasks while observers watch, listen and takes notes.  The goal is to identify any usability problems, collect qualitative and quantitative data and determine the participant's satisfaction with the product.” 3.

Ultimately, it is important for us to ensure the program meets the expectations of the people using it. We’ll heavily rely on our requirements traceability matrix to align with here.

## System Testing

System testing will allow us to test interactions with the application, and all aspects outside of the actual code itself. This is similar to usability testing in that it ensures the piece of software operates how a user or stakeholder expects it to operate, but in a much more literal way.

Another thing that can be said about system testing is that it tests your program ‘as a whole’. Rule99 describes this as “the testing of a complete and fully integrated software product. Usually software is only one element of a larger computer based system.” 4

# Characteristics of Test Plans

The characteristics of a test plan can help ensure a bulletproof testing process. This involves several stages, and specific techniques. Some specifics are included below:

* For deliverables we’ll be looking for
  + Test cases
  + Test plan
  + Negative testing (checking exception, error handling)
  + Potential testing scripts or algorithms (outside of the unit testing we plan on performing)
* Responsibilities
* Schedule
* Time estimation

A firm outline of the test plan allows us to be cognizant of our own time, and resources. This benefits both the stakeholders and the developers. An informed plan can help a testing process perform smoothly.

# Test Planning Approach

## Black Box Testing:

**Boundary Value Analysis (BVA)** will be used to test the outer limits of what we allow for input in Metric. This will be particularly useful for our application as we are loading in many different types of values such as percentages, 10s, 100s, and numbers in the thousands. BVA allows us to ensure extreme scenarios will not break or cause an inconsistent experience in the application.

## White Box Testing:

Our main focus here will be **Unit Testing** and **Integration Testing** to ensure the quality of our source code. Unit testing will allow us to fine tune and bulletproof our source code a single module at a time. Integration testing will ensure that the interfaces and integrated systems / libraries are not only connected and functioning, but are performing all of the needed functions seamlessly.

## Summary:

Much of our testing will be automated including the unit testing, and boundary Value Analysis. Our Integration testing will have a bit of hands on involvement as well as the setup of our unit testing.

Our overall goal here is to ensure a quality product upon release.

## 

# Outline for the Test Plan

## Scope (identification, system overview)

Our scope will include the feature set detailed in the requirement traceability matrix placed previously:

* File selection and navigation from local user’s computer
* Spreadsheet reading and writing.
* Spreadsheet editing
* User input
* Interface actions (button presses, text input action events)

## System description and operation (name of test sites, test configuration)

Our system testing will be performed on a staging site not publicly accessible. The web application will be accessed via <https://www.metrictestsite.com>. The site will have a webserver comprehending Java servlets.

Code structure will include two classes to be tested initially:

* Runnable interface class
  + This will include all GUI objects, input fields, and buttons
  + The source code will include a public static void main() method.
* Spreadsheet reading and writing class
  + This will house methods for file choosing, reading, and writing to spreadsheets using Apache POI libraries
  + The methods established here will be leveraged in the runnable interface class.

## Test identification & test levels

* Test ID will adhere to the document-naming scheme, CS459 IP.
* Test levels include:
  + Master Plan Testing (Comprehensive)
  + Black Box Testing
    - Phase Test Plan: Black Box
  + White Box Testing
    - Phase Test Plan: White Box

## Planned tests & test schedules

- Week 1:

- Black Box: BVA boundary testing for text input

- Black Box: GUI Action Event Testing

- White Box: Unit Test Setup

- Week 2:

- White Box: Unit Test Interface

- White Box: Branch Testing

- Week 3:

- White Box: Unit Test Spreadsheet Methods

- Black Box: GUI / BVA Testing

## Requirement traceability

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Requirements Traceability Matrix: Metric** |  |  |  |  |  |
|  | | | | | |
| **Project Name** | | **Metric** | **Business Area** | **QA Testing** | |
| **Project Manager** | | **Zach Sharpe** |  |  |  |
| **QA Lead** | | **Zach Sharpe** |  |  |  |
| **BR#** | **Category/Functional Activity** | **Requirement Description** | **Use Case Reference** | **Code Module/ Reference** | **Comments** |
| 1 | GUI Input | The software must be able to take in multiple fields and columns of a spreadsheet and parse the data into local data structures. | Entering Data | Spreadsheet Functions Class / readFile() Method |  |
| 2 | Display Output | It must be able to display this data back after parsing. | Data reporting for review | Spreadsheet Functions Class / showFile() Method | This refers to raw data |
| 3 | System Process | It must calculate averages and display them. Averages will not need to be performed for data averages that already exist. | Averages for goal setting / progress tracking. | Spreadsheet Functions Class / averageData() Method |  |
| 4 | Display Output | The data must be displayed in a visually, friendly manner. Ideally, a GUI will be used to display text to the user. | Data reporting for review | Interface Class / display() Method | GUI view of data will be different than raw data or spreadsheet information. |
| 5 | Systematic Output | Export to spreadsheet | Portability, data manipulation | Spreadsheet Functions Class / exportData() Method |  |
| 6 | GUI Input | Edit existing spreadsheet | Use GUI to edit an existing spreadsheet | Interface Class / getText() Method |  |
| 7 | GUI Input | Select file to edit | Edit existing a file | Interface Class / fileChooser() Method |  |
| 8 | Systematic Output | Select file to export over / update | Overwrite select fie | Interface Class / fileChooser() Method |  |
| 9 | Systematic Output | Name export file | Select file name for exported file | Interface Class / fileChooser() Method |  |
| 10 | System Process | Parse Data / Data Structure Loading | Store data | Spreadsheet Functions Class / ArrayList + Array Classes | Simple Arrays embedded into list objects. |
| 11 | GUI Input | Close Program | Shut the program down | Interface Class / System.close() Method |  |
| 12 | GUI Input / System Process | User Authentication | Authenticate user's entering application to protect import / export data. |  | Relying on SSO initially for authentication. |
| 13 | Systematic Output | Data Graphing | Provide different visual forms to process data | Interface Class / display() Method |  |
| 14 | System Process | Web application hosting | Providing a web application server to create an enviroment for the application to run. | Web Server Class / Connection objects | Apache will be used initially for hosting. |
| 15 | System Process | Data Storing | Server to host data from import / exports | mySQL server | AWS S3 considered for subsequent phases of development. Transition will be considered in design. |
| 16 | System Process | Past import data retrieval | View past imports to reflect on, manipulate, or update. | Class Interface / JDBC Methods |  |
| 17 | System Process | Optional past data purging | The ability to completely purge personal data in compliance with GDPR | mySQL Server |  |
| 18 | GUI Input | Select data reporting type | Choose how data gets graphed and displayed (point, bar, line graphing) | Class Interface / jButton Action Listener |  |
| 19 | System Process | Auto Saving | Store past input data periodically to prevent accidental data loss | Web Server Class | This may be a marriage between our mySQL store and web server capabilities |
| 20 | System Process | Save reporting / Export reports | Save image or spreadsheet data with visual reporting of Metric data. | Web Server Class | Similar process to exporting (BR 5) but visual representation. |

# Regression Testing

## Why Regression Testing

Regression testing can be a very subtle key to very successful software. To put simply I will borrow from Smartbear.com’s explanation of regression testing: “Whenever developers change or modify their software, even a small tweak can have unexpected consequences. Regression testing is testing existing software applications to make sure that a change or addition hasn’t broken any existing functionality.” 5

Because regression testing is a retrospective test method, it is continuously giving us confidence in the product that we are creating. This is the practice of reusing test methods that have been setup prior, creating new tests that check for aspects of the software that should now be considered, and even testing non source code related areas and how it affects the code base.

## Potential Impact Requiring Regression Testing

Small changes can have a large affect on the execution of code. No matter how big or how small a change to the source code, we will be following every phase with regression testing. To give some examples of impact from not regression testing, I’ve made a list of commonly missed aspects:

* Data Unavailability
  + This can be caused from database connections, changed SQL queries, or any structural change to the code base.
* Input Issues
  + Even despite rigorous black box testing, small changes to a code base can cause the user experience to change or be vulnerable to fringe exceptions that were previously not exposed.
  + Specifically, we will use smoke testing for many input tests and boundary tests.
* Testing old reports for new issues.
  + Relying on the results of older tests can allow a false sense of security within the application and eventually cause problems.

## How We Use Regression Testing

* Past Tests:
  + We’ll use various tests that have already been setup, while still examining if this test is setup to truly test the integrity of the *whole* module. This is large piece of efficiency that will be leveraged.
  + New tests will also be made. With new modules, new features, and new code come new dependencies. Tests are to be run on new modules, and new tests made for old modules to test for new aspects of code.
  + Black box testing will remain rigorous throughout major stages of implementation for the project. Similar to how we’ll treat unit testing, black box testing, and user testing, will be repeated through development.

# Revision Log for Unit 4

* 4/28/18 Added Regression testing sections:
  + Why Regression Testing
  + Potential Impact Requiring Regression Testing
  + How We Use Regression Testing
* 5/1/18 Added Raw RTM data instead of Excel Sheet Object for better visibility.
  + The RTM template used does not lend itself to embedding and should eventually be changed or linked to instead.
* 5/2/18 Added additional context to working environment details
  + Added context to physical work environments for better understanding.
  + Added context to ‘job requirement’ points to better communicate why we are looking for these requirements, instead of being purely informational.

Thank you to Samantha Kinsley for reviewing week 3 of this draft and providing invaluable feedback that was implemented for week 4!

# Summary

* **Final regression testing has been completed for launch. This has included the following:**
  + Regression testing all unit tests to check for changed dependencies.
  + Final black box testing for user experience.
  + Final bounds testing including index stretching unit tests and black box bounds testing.
* **Implemented all sources of feedback retroactively to the applied sections:**
  + Requirements Traceability Matrix (noted in change log)
  + Work environment
  + Overall formatting
  + Removal of Appendix A and addition to full body of documentation.
  + Updated table of contents

# References

1 P. (n.d.). Software testing traceability matrix templates. Retrieved April 03, 2018, from https://strongqa.com/qa-portal/testing-docs-templates/traceability-matrix

**2** Tarlinder, A. (2017). *Developer testing: Building quality into software*. Boston: Addison-Wesley.

3 Affairs, A. S. (2013, November 13). Usability Testing. Retrieved April 16, 2018, from https://www.usability.gov/how-to-and-tools/methods/usability-testing.html

4 What is System Testing? Types & Definition with Example. (n.d.). Retrieved April 17, 2018, from https://www.guru99.com/system-testing.html

5 What Is Regression Testing? (n.d.). Retrieved from https://smartbear.com/learn/automated-testing/what-is-regression-testing/