# SID – The Smart Investment Dashboard

**C. Malcolm Todd – T00232792**
**Saloni Saluja – T00608615**

# TESTING PROCEDURES

**PREPARED FOR:**

**Mr. Kevin O'Neil, TRU Computing Science**
**koneil@tru.ca**

# CONTENTS

# EXECUTIVE SUMMARY

**This report serves to define the testing procedures and tools to be employed throughout the development of SID, the Smart Investment Dashboard. For investors that use a self-directed investment strategy, SID will function as a resource by helping to mitigate or remove that strategy's current disadvantages. In developing SID, we will utilize an Agile software development process.**

**The development of SID will implement a comprehensive testing approach which uses a combination of Test-Driven Development, unit tests, integration tests, system tests, and release tests. By conducting an extensive combination of testing varieties, SID will provide improved discovery of defects, better quality coding, and reduced development costs. Testing of SID will also employ test automation to improve SID's regression testing, and to improve the amount of test coverage completed during each incremental release.**

**To implement our testing strategy, we will make use of an established testing framework that provides support for multiple programming languages and software architectures. Through the use of such a framework, we can create consistency in our testing and validation efforts regardless of any structural or development design choices made during SID's development. With this framework goal in mind, we have selected a framework named 'Allure' which provides the needed resources and useful reporting mechanisms.**

# VALIDATING OUR SOFTWARE

The primary goal of software validation is to establish confidence that SID is fit for the purpose that it was originally designed for [1]. For this reason, we will employ a comprehensive testing strategy consisting several types of tests throughout SID's development and evolution.

To provide for regression testing, these tests will also be repeated at each release to validate and maintain the required confidence in SID throughout its evolution. In order to make this repetition possible, we will rely on automated testing procedures in as many tests as possible. The types of software tests to be complete can be broken into Test Driven Development (TDD) for unit tests, integration and system tests, and acceptance and release testing in an Agile context [1].

## TDD and Unit Tests

A key characteristic to effective software testing and validation is the ability to test, and in turn validate, each of the system's user requirements [1]. Because of this critical need for clear requirements with which to test against, the development of SID will begin with Test Driven Development (TDD). Some of the key features and strengths of TDD are:

- TDD is common practice in many Agile methodologies where a software's unit tests are written before any coding starts [1]
- When coding to pass the created unit, the developer seeks to write "just enough" code to pass the test and to refactor the code until it conforms to a simplicity criteria [2]
- Using TDD creates a detailed roadmap describing an ideal version of the software [3]
- TDD serves to help expand and clarify the requirements of the system by identifying missing or vague requirements for which tests require additional information to create [3]
- Many teams that employ TDD report significant reduction in defect rates [2]

Using the battery of unit tests that are generated through the TDD process enables our team to test each of SID's components in isolation [1]. Testing in isolation has been shown to improve code quality, and allow developers to find defects earlier and more easily. These development benefits also lead to reduced development costs which also benefits the customer [4].

Our testing efforts will be further enhanced by automation. By automating, our team gains the ability to increase our test coverage which allows for more thorough testing at each incremental release [5]. Automation also reduces the need for manual testing, and will save our developers additional time in the long run to further reduce SID's development costs [5].

## System Tests

An additional layer to our testing strategy will include integration testing. After successfully conducting unit tests of each component, the next step will be to conduct integration and system testing. Integration and system tests both involve testing the interactions between components to determine their compatibility with each other [1].

Integration tests involves the grouping of components so that they may be tested together [6]. The purpose of integration testing is to test the interactions between each component which includes the use of their interfaces [1].

System tests have common elements to integration tests; however, system testing involves the integration of all components into a complete system which is tested [7]. The principle goal of system testing is to determine that the software meets all specified requirements [7] which makes it a test step in software validation. System testing often involves a separate testing team to conduct the test [1]; however, given our organizations size, we will complete these tests with same team which consists of our developers and a customer representative.

## Release & Acceptance Tests

After successfully testing SID as a system, many software projects move on to the release and acceptance testing phase. Release testing is a form of system testing which is designed to be run outside of the development team and follows a Black Box testing process [1]. With Black Box testing, the internal structure and implementation is hidden from the tester allowing for test cases that do not factor in these internal aspects of the system [8].

In Agile, there is typically no separate phase for acceptance testing. This is due to the customer's involvement as part of the agile process which allows for integration of tests created by the user into the other testing phasing [1]. By having the customer, as a content expert, means that Agile typically seeks to complete acceptance testing during each current sprint just before the deployment takes place [9].

Having this phase of testing built into our Agile process is also helpful to our organization as it will allow us to automate some of these test cases, rather than solely relying on manual user testing in the field for all acceptance test cases. By being automated, this also allows for some of SID's acceptance testing to form part of our regression test suite.

# OUR CHOICE OF TESTING TOOLS

## What is Allure?

Allure Framework is a flexible lightweight test report tool that supports multiple languages such as Java, Python, JavaScript, Ruby, Groovy, PHP, .Net, and Scala. It shows a concise representation of the test in an organized web report form [10], thereby allowing us to extract maximum useful information from the everyday execution of tests.

As from the Dashboard Developer's point of view, Allure reports would benefit us by cutting down the typical lifecycle followed by defects: test failures can be divided on bugs and broken tests, also logs, steps, fixtures, attachments, timings, history and integrations with TMS and bug-tracking systems can be configured [10]. This enable us to have all key information on hand, which allows us to focus on any critical validation areas.

From business perspective, Allure would provide us with a big picture of the features that have been covered, defects that are clustered, the timeline of execution, and many other useful graphs and statistics [10]. Allure's modularity and extensibility will prove incredibly valuable over the course of the development of SID, due to its ability to fit with any functional change as SID evolves over time.

## Working with Allure

An appealing aspect of working with Allure is its simplicity of use. This is due to the fact that there are only two steps involved in the generation of a test report.

The first step is the test execution, in which a small library called the 'Adapter' is attached to the testing framework, and is responsible for saving information about tests that have been executed along with the XML files [11]. Allure provides a series of Adapters for Java, PHP, Ruby, Python, Scala, and C# testing frameworks.

The second step is the report generation, in which XML files are transformed to an HTML report for viewing. Allure is based on standard results output by xUnit, but adds some supplementary data. There are several ways for generating these reports, including command line tools, a plugin for CI, or a build tool [11].

# Reporting Format

**A typical report consists of several tabs for different kinds of test data representation, and test case pages for each individual test. Each Allure report is backed by tree-like structures that represents a test execution process. These tree-like representations include Behaviors, Categories, xUnit and Packages [10]. All of these trees support filtering and sorting, which is of benefit to us as it provides an easy mechanism for isolating key test data. Different tabs would allow us to switch between the views of each original data structure [10], thus giving us a range of different perspectives.**

## Overview Page

**The Overview Page shows the overall test execution statistics with a list of parameters used for testing. The section on the right contains a list of the top defects found in the software. These defects are grouped by defect messages [11].**

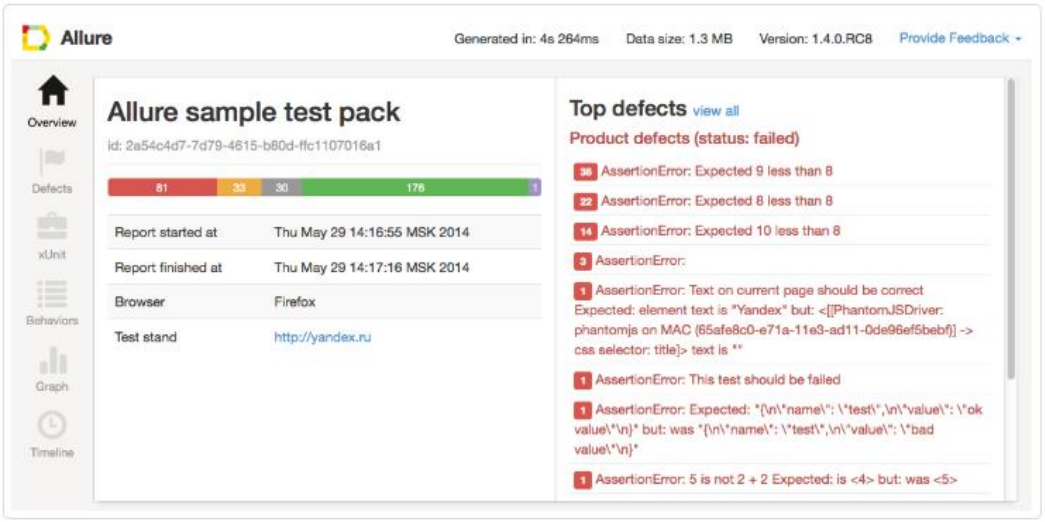


**Figure 1: xUnit, Source: Adapted from [11]**

## Defects Page

**The defects page provides a detailed list of defects that are revealed at the time of execution. Products Defects correspond to failed tests and Test Defects correspond to broken tests [11].**

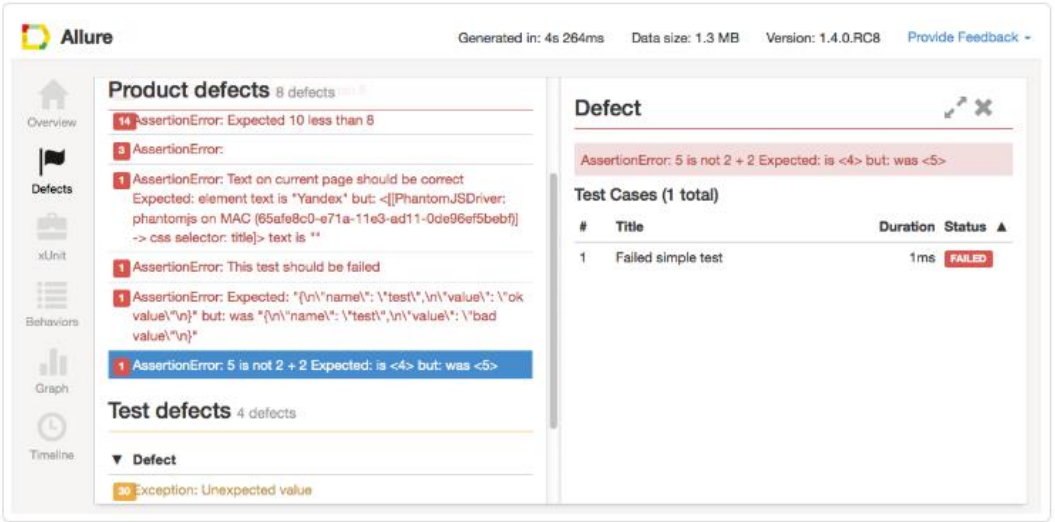


**Figure 2: xUnit, Source: Adapted from [11]**

# ALLURE REPORTS (CONT)

## xUnit Page

The xUnit page shows statistics in terms of xUnit. It is possible to view test statistics for each test suite and detailed information about every test case [11]. Using this screen, we can easily view which features and stories have problems.

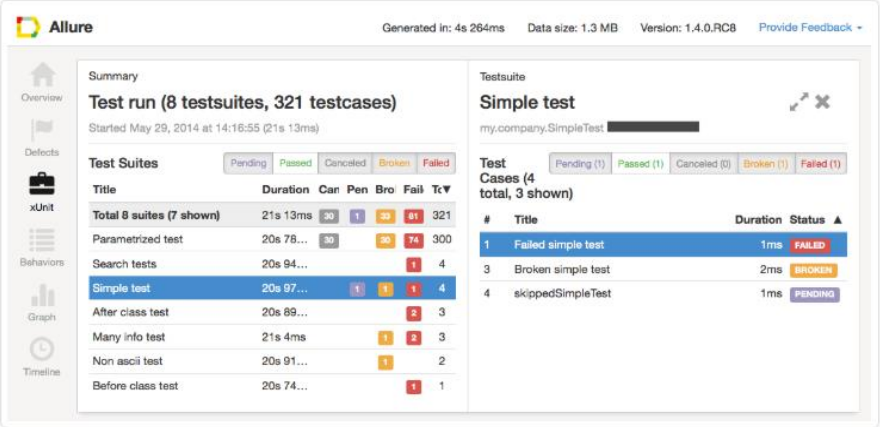


Figure 3: xUnit, Source: Adapted from [11]

## Graphs Page

The Graphs page allows us to see different statistics collected from the test data, the status breakdown of severity, or the duration diagrams [11]. Status diagrams help us identify the test pass percentage. Severity diagrams depict the severity of failure. Duration diagrams tell us about the total running duration.

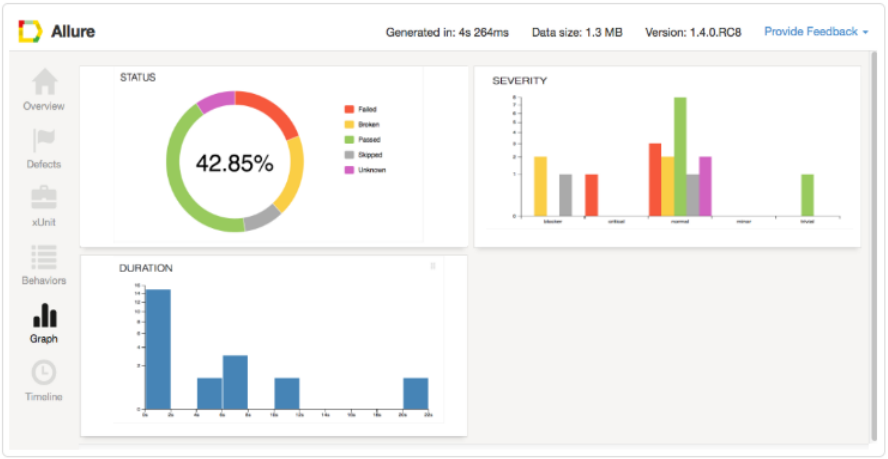


Figure 4: Graphs, Source: Adapted from [11]

## Timeline Page

The timeline tab shows the point of time at which each test case began execution, the total running time, and time at which a test failed [11]. This view will prove valuable to allowing us to analyze the efficiency of the involved components.


Figure 5: Timeline, Source: Adapted from [11]

# CONCLUSION

In conclusion, the development of SID will employ a thorough software validation process. To conduct this thorough strategy, we will utilize Test Driven Development (TDD), unit testing, system testing, and release testing. Our testing strategy will also employ test automation wherever possible as this improves the ability to test SID more regularly and thoroughly by repeating all automated tests during each incremental release.

By employing Agile, we will make use of TDD to generate a battery of unit tests. After successfully testing our components in isolation, we will then proceed with integration and system tests. Agile methods also complete their acceptance testing prior to deployment [9], which will allow us to automate and include certain of these tests in our regression test suite.

To provide for implementation of our testing process, we require a flexible lightweight tool that is capable of supporting multiple programming languages. Using this type of framework will ensure consistency regardless of what programming language or software architecture design decisions that will occur during SID's development and evolution.

To meet our testing framework needs, we selected the 'Allure' framework that provides the required flexibility. Allure also provides the means to implement our goal of test automation and a simplified reporting format [10] that makes it very attractive choice. Allure's reporting provides well organized perspectives such as an overview page, defect page, summary page, graph page, and timeline page [10]. With pertinent reporting, the ability to automate SID's software tests, and Allure's support for multiple programming languages make it the ideal framework for our purposes.

# REFERENCES


[1] I. Sommerville, Software Engineering (10th Edition), Pearson, 2015.

[2] Agile Alliance, "TDD," Agile Alliance, 2020. [Online]. Available: https://www.agilealliance.org/glossary/tdd/

[3] A. Strong, "Why Use Test-Driven Development?," Codecademy.com, 22 January 2018. [Online]. Available: https://news.codecademy.com/test-driven-development/. [Accessed 28 January 2020].

[4] E. Novoseltseva, "dzone.com," Devada Media Property, 30 August 2019. [Online]. Available: https://dzone.com/articles/top-8-benefits-of-unit-testing. [Accessed 29 January 2020].

[5] SmartBear Software, "What Is The Benefit of Test Automation and Why Should We Do It?," SmartBear Software, 2020. [Online]. Available: https://smartbear.com/solutions/automated-testing/. [Accessed 29 January 2020].

[6] "Integration Testing," Software Testing Fundamentals, 2019. [Online]. Available: http://softwaretestingfundamentals.com/integration-testing/. [Accessed 30 January 2020].

[7] "System Testing," Software Testing Fundamentals, 2019. [Online]. Available: http://softwaretestingfundamentals.com/system-testing/. [Accessed 30 January 2020].

[8] Software Testing Fundamentals, "Black Box Testing," Software Testing Fundamentals, 2019. [Online]. Available: http://softwaretestingfundamentals.com/black-box-testing/. [Accessed 30 January 2020].

[9] 360Logica.com, "How to Perform User Acceptance Testing Using an Agile Process?," 360Logica.com, 28 May 2018. [Online]. Available: https://www.360logica.com/blog/how-perform-user-acceptance-testing-using-agile-process/. [Accessed 30 January 2020].

[10] Qameta Software Team, "Allure Framework," Qameta Software Team, 19 September 2019. [Online]. Available: https://docs.qameta.io/allure/#_about. [Accessed 29 January 2020].

[11] Qameta Software Team, "Allure Test Report," Qameta Software Team, 19 September 2019. [Online]. Available: http://allure.qatools.ru/. [Accessed 29 January 2020].

[12] M. Staron, "Dashboard development guide," Chalmers University of Technology and University of Gothenburg, Göteborg, Sweden, 2015.

[13] C.-E. Lavoie, "How to Build a Dashboard Using Agile Methodology [Part 1]," agileDSS, July 2013. [Online]. Available: https://www.agiledss.com/en/blog/how-build-dashboard-using-agile-methodology-part-1. [Accessed 19 January 2020].