**Junit Test Runner**

**Software Detailed Design**

**CEN 4072 Software Testing, Spring, 2015**

Github Repository: <https://github.com/gbkuhn/Junit_Test_Runner.git>

**Modification history:**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Who | Comment |
| v1.0 | Mar. 7, 2015 | Geoffrey Kuhn | Initial Detailed Design |
| V1.1 | Mar. 12, 2015 | Carly Hessler | Revisions |
| V1.2 | Mar. 12 2015 | Geoffrey Kuhn | Final edits |
| V1.3 | Mar. 18, 2015 | Geoffrey Kuhn | Revisions to integration, classes and dependencies |
| V1.4 | Mar. 28 2015 | Geoffrey Kuhn | Updated class diagram, improved program structure |
| V1.5 | Mar. 31, 2015 | Carly Hessler | Updated Class Diagram |

**Team Name:** Kuhn-Hessler

**Team Members:**

* Geoffrey Kuhn, gbkuhn@eagle.fgcu.edu
* Carly Hessler, cahessler3098@eagle.fgcu.edu

**Contents of this Document**

[Interfaces and Classes](#Section1)

[Class Diagram](#Section2)

[Dependencies](#Section3)

[Unit Test Plan](#Section4)

[Integration Test Plan](#Section5)

**SECTION 1: Interfaces and Classes**

Interfaces

* Junit
* Desired
* Runner\_info

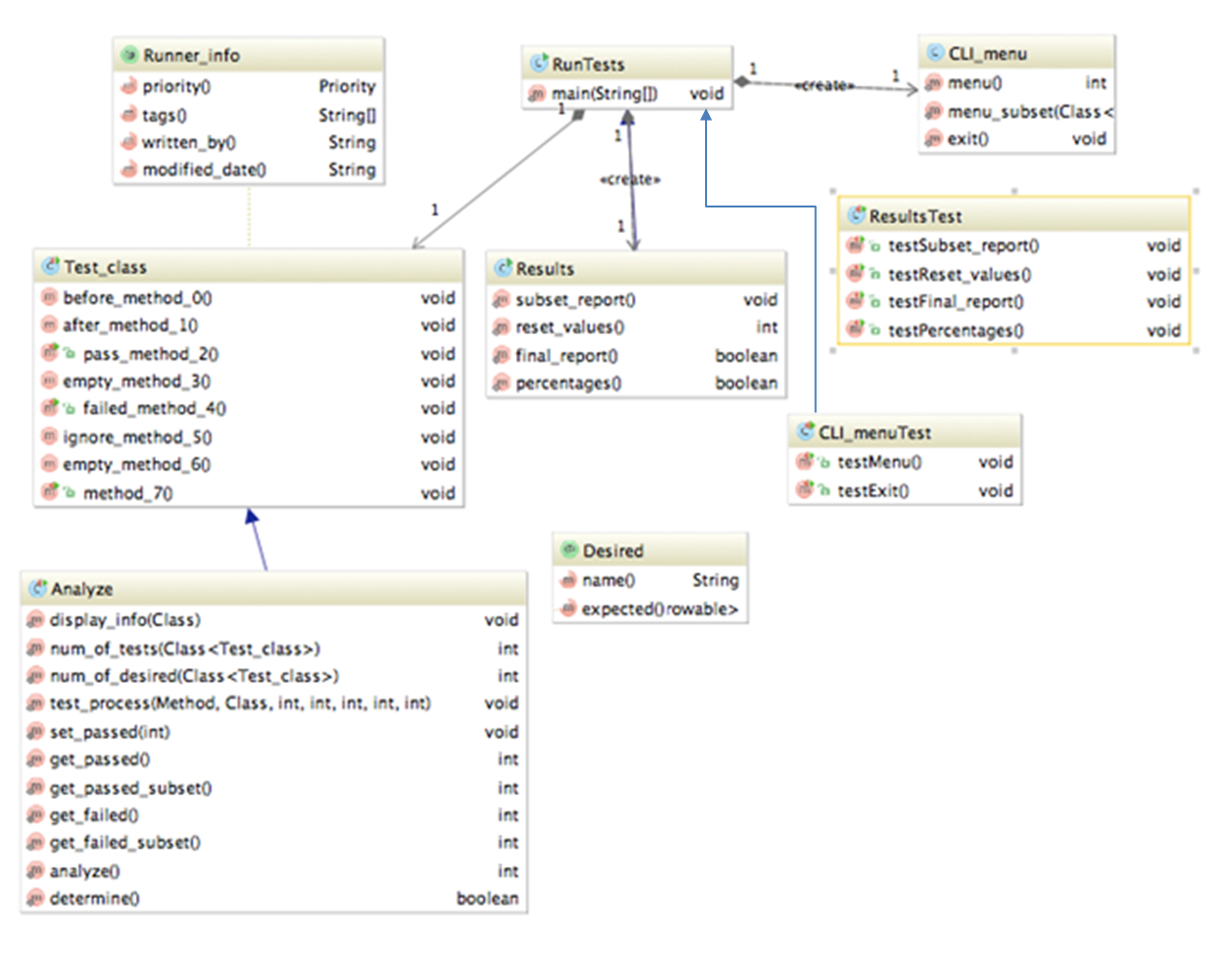
Classes

* Analytics
  + Analyze
  + RunTests
* Menu functionality
  + CLI\_menu
  + Results
* Testing
  + Test\_class

ResultsTest

CLI\_menuTest

**SECTION 2: Class Diagram**



**Runner\_info:** The “Runner\_info” interface provides a description of the software being developed. Such as the programmer and the date modified. If this interface is found it displays such information.

**RunTests:** Holds the main method. Iterates through the test cases.

**CLI\_menu:** Provides the user prompts through the command line, such as the amount of times a certain percentage of tests should run. These values are designated by the user.

**Test\_class:** This class holds Junit Test cases for the testing the functionality of the software

**Results:** Manages the values for the amount and types of test cases. Has functionality to print the results at the end of the entire run and the subset report for the end of each subset run.

**ResultsTest**: Test coverage for the Results class.

**Analyze:** Manages the @Desired test annotation and provides setters and getters for subset test values.

**Desired:** This interface provides the annotation to have such a test case priority over others.

**CLI\_menuTest:** Test coverage for menu.

**SECTION 3: Dependencies**

Junit as a dependency is used to create the custom annotation as needed. Below, a new interface is created for one of the new annotations called priority, this labels the test method as important for the runtime.

The Junit test runner suite will run. The implementation will use Junit and Maven to configure the dependencies as follows:

Dependency Declaration

Below is the dependency declaration for the Desired annotation. It is executed on runtime when the runner comes across the annotation.

**import** java.lang.annotation.ElementType;  
**import** java.lang.annotation.Retention;  
**import** java.lang.annotation.RetentionPolicy;  
**import** java.lang.annotation.Target;

@Retention(RetentionPolicy.***RUNTIME***)  
@Target(ElementType.***METHOD***)  
**public** @**interface** Desired {  
 String name() **default ""**;Class<? **extends** Throwable> expected() **default** None.**class**;  
 **static class** None **extends** Throwable {  
 }  
}

//implementing custom annotation

1 @Target(ElementTpe.METHOD)

2 @Retention(RetentionPolicy.RUNTIME)

3 public @interface D{

4 Class example();

5 //make method a priority for testing

6 }

Dependency Usage

//usage for custom annotation

1 @Desired

2 public static void priority\_method{

3 //runs test if a priority

4 }

What is not included, is the parser for seeing the custom annotation on the Github repository.

**SECTION 4: Unit Test Plan**

The “Results” class will include all of the results of the tests from the test runner and the performance statistics of the test runner. Including the amount of methods that ran given a subset. These values will be compared with values designated by our group in separate test cases using Junit to test its success. The idea of using a test runner to test others tests is the idea being utilized.

The classes with will be unit tested will be Test\_class and Results. The following classes will compose the entire project.

* Analyze.java
* RunTests.java
* CLI\_menu.java
* Results.java
* Test\_class.java
* Priority.java
* PriorityAnnotationParse.java
* Desired
* Runner\_info
* ResultsTest.java

A coverage of %80 is expected for this this project with a goal of at least %90.

**SECTION 5: Integration Test Plan**

The integration test plan is to test the components of the test runner without a front end such as an interface. The first integration tests will be utilizing the custom Junit annotations such as “@Priority” and “@Low\_priority” with the existing annotation, such as “@Before”, “@After”, “@Test” etc.

The next portion of the integration testing will be dividing the tests into subsets defined by the users. These will be initially implemented by being done inside the program itself. For example, if the user selects only three classes to run, only those three classes will run with associated test annotations.

For testing the functionally of all of the components, the expected results in the Results class will be compared with expected values using Junit assertions. The final component is to implement the parameters for the test runner using a command line interface. At least 80% of methods will be covered by tests.

The following classes will be tested with its dependencies in the integration test:

* Results.java
* RunTests.java
* Test\_class.java

Template created by G. Walton ([GWalton@mail.ucf.edu](mailto:GWalton@mail.ucf.edu)) on Aug 30, 1999 and last updated Aug 15, 2000; updated by A. Koufakou, Aug 2014, updated by D. Guo, Feb 2015

This page last modified by Dahai Guo (dguo@fgcu.edu ) on Feb 9, 2015