Team Name: Test Bayes
Team Members:

Adavya Bhalla (bhalla)
Aditya Jhamb (aditya97)
Avidant Bhagat (avidant)
Ethan Mayer (emayer4)
Steven Austin (saaustin)

Architecture and Implementation:

Interaction:

JUnit is an annotation based test system that can easily run large test suites by providing a convenient framework for developers to use. JUnit gives a test suite class that a developer can extend, and fill with test methods, that can then be easily run with a simple command (or simple GUI interaction). JUnit essentially is an API that a developer can use to wrap around there tests to make development quicker. However, JUnit works with a static ordering of tests, which does not allow for tests that are most likely to fail to actually fail first, causing for a lot of wasted time.

Our system will not have a GUI, as it will be completely command line based, so as to be used by all systems, instead of just those that have GUIs. Our design will interface with JUnit as an extension, meaning we will not modify JUnit in any way, and will simply act as a wrapper for JUnit, interfacing JUnit in a way similar to how a user would interface with JUnit. We will run tests one by one, in the order that we have determined mathematically.

The user will be able to interface our design through a very similar way to JUnit. They will be able to run all tests, run specific tests, and run a certain group of tests. Our design also creates a notion of a failure probability difference threshold (epsilon), and a moving average amount. When given a scenario where two tests have roughly the same failure probability (difference less than or equal to epsilon), we will run the shorter running test, instead of the longer one, so that the shorter one can fail first. This makes our program run not only the most likely to fail, but the shortest tests that are most likely to fail. Our moving average is used to not incorporate test runs that are too old, as tests often change while the developer is in the process of debugging. This moving average can be tuned (increased or decreased) to incorporate more or less most recent test runs in the failure probability calculations. The user can also choose to not reorder tests, just for sanity purposes. All of these configurations will be modified through the pom configuration file in the maven project that our design will be a part of, since our design will use maven as its build system.

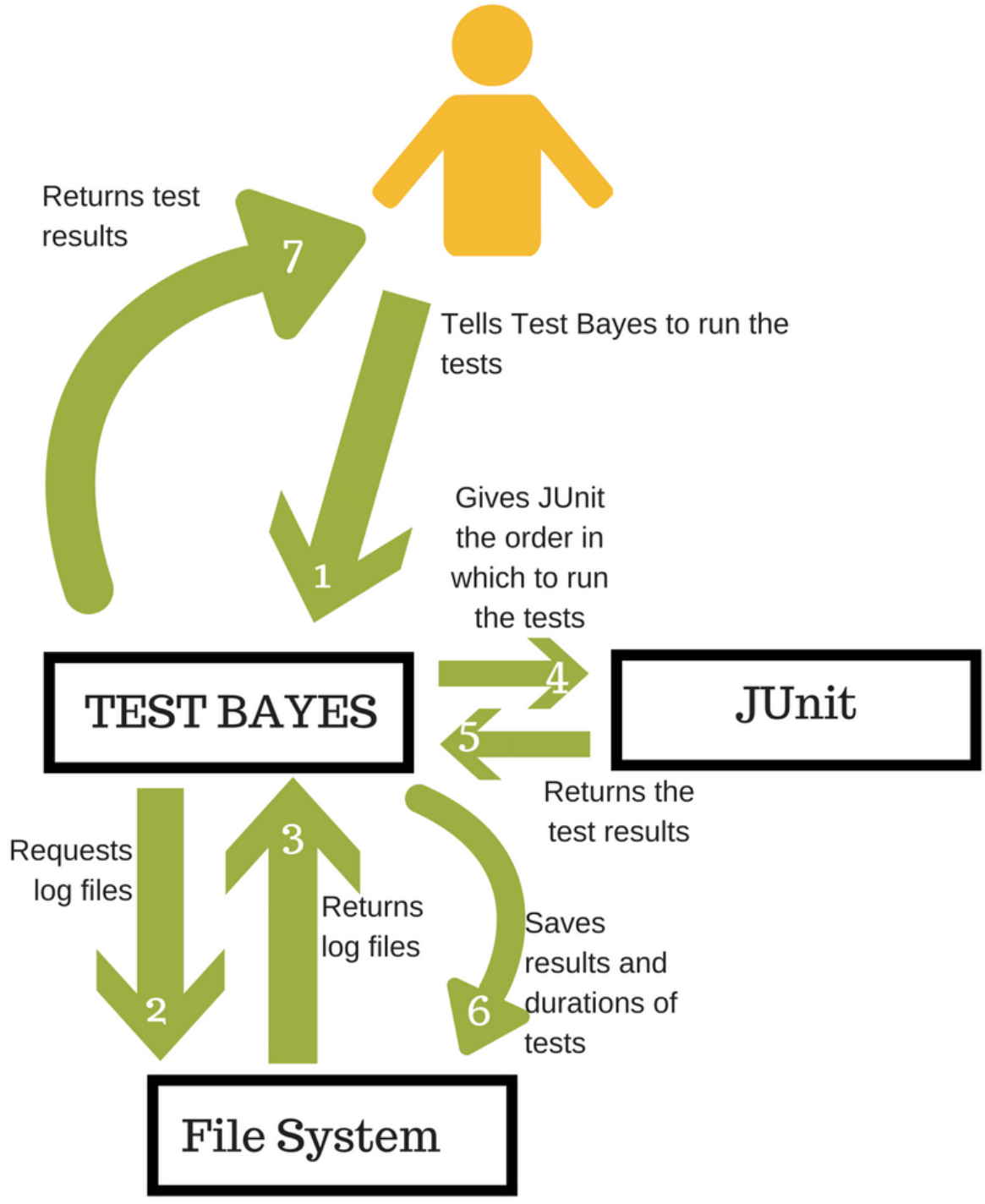
These changes will not affect the JUnit architecture, as we are only acting as an extension of JUnit, not actually modifying JUnit itself, leaving JUnit to be a tool that our design uses under the hood.

Implementation:

We have decided to make 2 implementations for the actual test ordering. It is not actually 2 completely different implementations, it is just the same code but ordered in a different way. The first implementation involves a greedy heuristic where we first run the test with the most probability of failing and then based on the outcome of the first test greedily decide which test to run next based on conditional probability. We would break ties on the greedy decisions by factoring in time taken to run the test, conditional probability, total probability, and a running average. The second implementation would be to just use everything from the start instead of waiting for ties. Evaluation will be critical in deciding which implementation will work better. Here are a few details about the data structures we plan on using:

1. **HashMap for total probability -** We want to store the total probability for each test passing in a HashMap where the hashCode is a code created by the name for each test and the value is the probability for each test passing. The value will be in a fraction object either created by us or an implementation we find online if it is able to match all the operations we would want to perform on it.
2. **HashMap for runtime -** We want to store the runtime for each test in a HashMap where the hashCode is a code created by the name for each test and the value is the runtime for the given test. The runtime will be stored as a double with seconds as the unit of time.
3. **Matrix for Conditional probability -** This is where we still have certain grey areas. We want to store the conditional probability in a matrix so that if we access the value for a test x by test y index we get the probability for test x passing given test y either passes or fails. There will be 2 of these matrices to facilitate faster calculations and decrease computing time. One way of doing this is to have a HashMap that itself has a HashMap as a value and then the second HashMap contains a fraction object as value which is the probability we seek. A 2D array implementation might also suffice. These are the 2 options we are currently looking into and right now we like the HashMap implementation better.

Architecture:



We will be extending junit, specifically the BlockJUnit4ClassRunner (the class which runs each test individually), in order to change the order in which tests are run. The uncertainty with the class runner is that it runs tests from one class at a time; we need to order all of the tests at once, not just the ones in a single class. Once the tests are ordered, they will be executed in a standard fashion by junit, but additional test results will be recorded to a file to be analyzed next run.

Evaluation:

To evaluate our code, we will be using two metrics to compare against two alternative ways of ordering tests. We will track both the time and number of tests run until the first failure, and compare our results for these figures to a test suite run in the default junit ordering, and a randomly ordered test suite. We will run these tests on multiple repositories, for example by injecting bugs into the CSE 331 RatPoly code, and compare how our ordering fairs on each execution. We expect that test Bayes ordering will improve dramatically after a few runs of the test suite.

Repository Names	Time taken to first failure			Number of tests to first failure		
	JUnit Ordering	Random Ordering	TestBayes Ordering	JUnit Ordering	Random Ordering	TestBayes Ordering

Resources Used:

http://www.baeldung.com/junit-4-custom-runners

http://junit.sourceforge.net/javadoc/org/junit/runners/BlockJUnit4ClassRunner.html

https://github.com/eugenp/tutorials/blob/776a01429eb6b7ec0d1153f88097e2cc9915e599/testing-modules/testing/src/test/java/com/baeldung/junit/BlockingTestRunner.java

https://stackoverflow.com/questions/20976101/how-to-get-a-collection-of-tests-in-a-junit-4-test-suite?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa

https://junit.org/junit4/javadoc/4.12/org/junit/runner/manipulation/Sortable.html

https://ieeexplore.ieee.org/abstract/document/1007961/

https://ieeexplore.ieee.org/abstract/document/5401169/

https://ieeexplore.ieee.org/abstract/document/1510150/