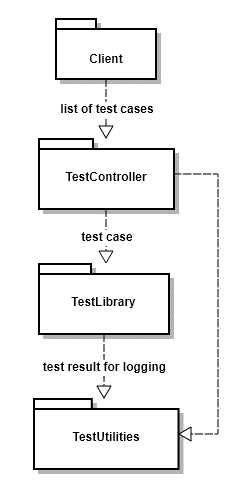
**Test Harness**

**Phase #2 – Architecture and Design**

**Group:** Santhosh Srinivasan, Jiawen Zhen, Alifa Stith

**1. Architecture**

The high-level architecture for Test Harness can be reasonably divided into three distinct components: *TestController*, *TestLibrary*, and *TestUtilities*. The *Client* component is included to represent an easily extensible input source (console, GUI, etc.).



**1.1. TestController Package**

The *TestController* package is the main entry point to the Test Harness application and will be responsible for the following functionality: (1) launching the Test Harness application and receiving input provided by the user; (2) sending each *TEST* function to be run by *TestLibrary*; (3) initializing and maintaining queues (future); and (4) storing and maintaining a summary of test results.

**1.2. TestLibrary Package**

The *TestLibrary* package can be thought of as providing the main functionality of the application. This is the package responsible for running each test individually, obtaining the result, and catching and handling any exceptions. The *TestLibrary* package also produces and formats the test result log messages as well as any exception messages.

**1.3. TestUtilities Package**

The job of the *TestUtilities* package is to collect and log the output data throughout the execution of Test Harness. It calculates time information for each test as well as for the summary of test results. This package is crucially responsible for writing the logging information to permanent storage within the file system.

Figure 1. Block diagram of major components of Test Harness, with arrows indicating the flow of program execution.

**1.4. Queue Interface**

Currently, all our packages communicate directly through function calls. However, we attempted to organize our code in such a way to be easily extended to support concurrency in the future. Particularly, we plan on implementing a queue to hold test results and log messages. The *TestLibrary* package will enqueue log messages onto the queue as it obtains test results, while the *TestUtilities* package dequeues and logs them. This interface will allow us to run *TEST* functions in parallel without causing I/O errors when logging results.

**2. Design**

The design of Test Harness breaks down the architecture from the previous section into individual classes. The *TestController* package is comprised of three classes: *TestHarness*, *TestIdentifier*, and *TestResultCounter*; the *TestLibrary* package contains an additional three classes: *TestRunner*, *TestAssertion*, and *TestExceptionHandler*; and two classes make up the *TestUtilities* package: *TestLogger* and *TestTimer*.

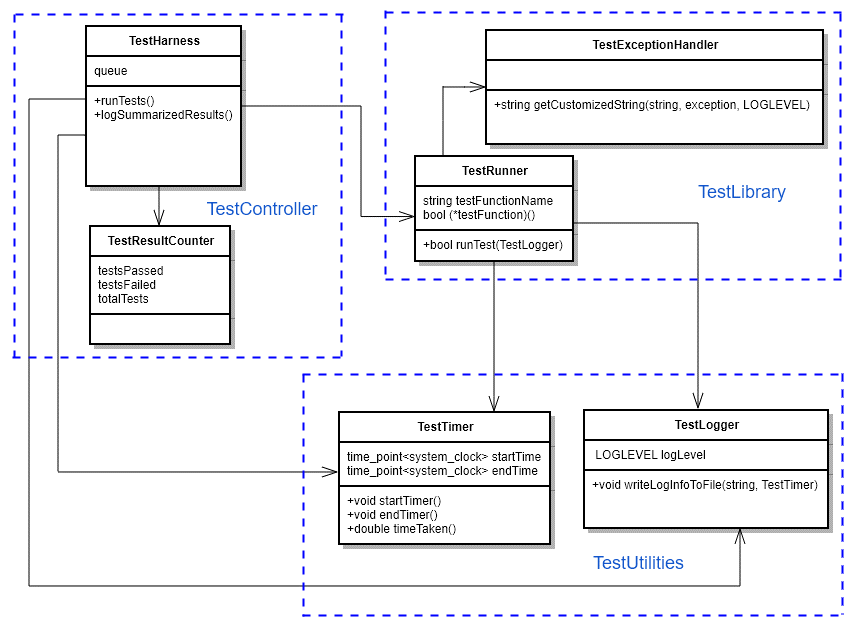


Figure 2. Class diagram (UML) for Test Harness. Classes are grouped by package membership, and arrows show USING relationships between classes.

**2.1. Classes in TestController Package:**

**TestHarness, TestIdentifier, TestResultCounter**

The *TestHarness* class is the entry point into the Test Harness application. Its first task is to obtain the *TEST* functions to test, which it does by prompting the *TestIdentifier* class to provide tests from the source code. *TestIdentifier* then scans the source code, collecting pointers to all *TEST* functions and enqueuing them onto Q1. *TestHarness* also initiates the *TestLibrary* package’s execution of tests on the functions in Q1. Finally, *TestHarness* provides a communication interface between the *TestResultCounter* class, which stores and maintains a summary of test results, and the classes of the *TestUtilities* package.

**2.2. Classes in TestLibrary Package:**

**TestRunner, TestAssertion, TestExceptionHandler**

Test Execution begins when the *TestHarness* class (in the *TestController* package) starts *TestRunner*, which is the entry point into the TestLibrary package. The *TestRunner* class is responsible for receiving (dequeuing) tests from Q1 to be run, collecting test results and exception messages, and enqueuing said messages onto Q2. *TestRunner* sends tests to *TestAssertion*, which contains the full library of testing methods in the Test Harness. The *TestAssertion* class runs its testing methods, and the *TestExceptionHandler* class catches any thrown exceptions and obtains the exception messages. Both the *TestAssertion* class and the *TestExceptionHandler* class provide their results to *TestRunner* for logging.

**2.3. Classes in TestUtilities Package:**

**TestLogger, TestTimer**

The *TestLogger* class will receive and dequeue logging and exception messages from Q2 and write them to the appropriate log file(s). It also will obtain and log the summarized results of the entire Test Harness session from the *TestController* package. The *TestTimer* class maintains and provides timestamp information for each test as well as the entire session summary. It is utilized by the *TestLibrary* package to time each test and set message timestamps, and by the *TestController* package to store intermediate data prior to logging the summary.

**3. Team Member Roles**

We have tentatively assigned ourselves to be responsible for one package each:

* Alifa – TestController
* Santhosh – TestLibrary
* Jiawen – TestUtilities

However, none of us intend to claim complete “ownership” of our assigned packages. Instead, we intend to collaborate heavily during the implementation phases of the project, especially throughout areas of the code that are more complex and time intensive. In phase 1 the team collaborated through Zoom calls to settle on the architecture and design of the project. The entire team was responsible for the architecture and design concept. A Github repository was created to share generated documentation and code produced during this project. A separate slack channel was created for team communications. Moving forward the team will use these shared resources to communicate and provide feedback for one another.