

# **TEST HARNESS**

## **Operational Concept Document**

Meka Ramakrishna Sayee

SUID: 993000736

CSE 681: Software Modelling and Analysis

Instructor: Professor Jim Fawcett

Date: Sep 19<sup>th</sup> 2016

## Contents

1. Executive Summary.....	3
2. Introduction .....	5
2.1 Objective and Key Idea .....	5
2.2 Obligations .....	6
2.3 Organizing Principles.....	7
2.4 High Level Architecture.....	7
3. Users and Uses.....	9
3.1 Actors .....	9
3.1.1 Project Developers .....	9
3.1.2 Project Managers .....	9
3.1.3 Quality Assurance .....	10
3.1.4 Teaching Assistants and Instructor .....	10
3.2 Uses.....	10
4. Partitions .....	11
4.1 Client Server.....	11
4.2 Repository Server.....	13
4.3 Test Executive .....	13
4.4 Test Queue .....	14
4.5 XML Parser .....	14
4.6 AppDomain .....	14
4.7 Loader .....	15
4.8 Test Logger .....	15
4.9 File Manager .....	15
4.10 Optional Modules .....	16
5. Activity Diagram.....	17
5.1 Activity Diagram for Loading DLL's in to the Repository .....	17
5.2 Activity Diagram for Testing the Code and Displaying the Results .....	19
6. Critical Issues and Analysis.....	21
7. Conclusion.....	23
8. Appendix .....	24
8.1 Message Creation and Parsing Package.....	24

8.2 Message passing communication package(s) that send messages between local clients and a remote server. ....	25
9.References .....	27

## Figures

Figure 1 Basic Activities of Test Harness .....	6
Figure 2: High Level Architecture of Test Harness System .....	8
Figure 3 Package Diagram for Test Harness.....	11
Figure 4: Activity Diagram for Loading DLL's in to the Repository .....	17
Figure 5: Activity Diagram for Testing the Code and Displaying the Results.....	19

## 1. Executive Summary

This document represents the Operational Concept document of Test Harness for the support of integration tests, which are the fundamental part of continuous software integration.

Many of the industries and organizations have adapted to the concept of unit and functional testing, but very few organizations focus on integration testing which is an integral part of software testing life cycle. Integration tests are mostly automated tests where a newly written project code is tested with the project baseline. Integration tests catches system level failures. There are several advantages to support integration testing in software testing life cycle such as,

- It becomes easy to identify and isolate modular level failures.
- These tests run in development environment which helps in identifying bugs in earlier phases of software development.
- Implores the developer to have a testing mindset for developing while developing test suite.

In general, test harness is a framework exercised in a system to automate the process integration testing. It reduces the load on the developer to perform testing, so that developer could focus on actual code which stimulates profitability for the organization. It ensures quality of the system at each stage of integration, when a new code gets added to the project baseline. Test harness enables test execution by using test libraries, test cases provided by the user and finally generates test reports that is stored in the database for future reference.

In this project, we are implementing test harness server which automates the process of integration testing. The system is implemented using C#, visual studio 2015 and the facilities provide by the .Net framework. The project should have the following specifications:

- Test harness should accept test requests form the client server and fetches the data from the repository server, where test code and test drivers are stored.
- The test code and drivers are loaded in to the repository by the client using a GUI. If a Test Request specifies test DLLs not available from the Repository, the Test Harness server will send back an error message to the client.

- Test Harness supports multiple clients to send test requests concurrently. The results for the test requests are also processed concurrently.
- To support concurrent access of multiple clients we use Threads.
- Test processing is done in AppDomain for each test request which separated from test harness.
- The unhandled exceptions should not interfere with the functioning of test harness.
- Test harness stores the results of each test requests for the respective client in the repository server in the form of file.
- The repository server should retrieve the test data and logs when requested by the user.
- All the communication between Clients, Test Harness and the Repository is implemented using Windows Communication Foundation (WCF).
- The Graphical User Interface (GUI) for the Client is constructed using Windows Presentation Foundation (WPF).

Developers, Quality Assurance, Project Manager, Teaching Assistants and Instructor are the primary users for which the system being implemented.

While implementing this system there are some critical issues which are to be considered. Further sections will discuss the solutions for the issues. Following are some of the major issues to be considered

- The system provided should be easy to use for the user.
- Maintaining the performance and load on the test harness.
- Access levels and authorization for various users viz. developers, quality assurance and project managers.
- Supports concurrent processing of test requests by multiple users.
- Maintaining version details of the project baseline.

In further section of the operational concept document we discuss high level architecture of the system, various modules in the system implementation, various diagrams associated with the system (viz. package diagram, activity diagram and flow charts), and critical issues. Thus, test harness system ensures that it supports continuous integration tests in an automated manner, which can be implemented in considerable amount of time and resources.

## 2. Introduction

Testing evaluates whether the component or the entire system as whole meets the requirements specified by the client. It helps in developing system which is error/defect free. In software development life cycle testing starts at early stages and continues throughout the life cycle. In developing big and real time systems, code has to be partitioned in to smaller chunks of data and thoroughly tested before inserting into project baseline. To perform this kind of testing we use test harness to run tests cheaply.

### 2.1 Objective and Key Idea

Test Harness is a tool we are implementing to automate the process of such testing, as it becomes difficult to test each and every chunk of code manually before adding in to project baseline. While developing large systems each module of code is developed by different developers and they should be able to access the project baseline simultaneously to test the new code that was developed with the project baseline before adding. To support this type of testing we are building a Test Harness Server where multiple clients can access concurrently to test their code before adding to the project baseline. The Test results are stored in a separate repository server to support client queries about the test performed.

Figure 1 describes the basic activities performed by the test harness system. The purpose of the diagram is to provide the readers an abstract idea about the set of activities the tool performs from the client's end.

As shown in the figure, the client first invokes GUI with which he can communicate with the Test Harness Server and Repository Server. In our implementation, we have multiple clients accessing the Test Harness and Repository Server respectively the following diagram the activities are mentioned for single client. Then the client loads the test code and test drivers to the Repository and pass the Test Requests in the form of XML to the Test Harness. If the Dynamic Link Libraries(DLL) for test code and driver are loaded successfully then the test harness accepts the test requests, parses the requests, performs the tests for the respective test code and stores the test results in the repository. Suppose, the Test DLL's are not present, the test harness sends error message to the client saying the Test code and Test Driver DLL's are missing.

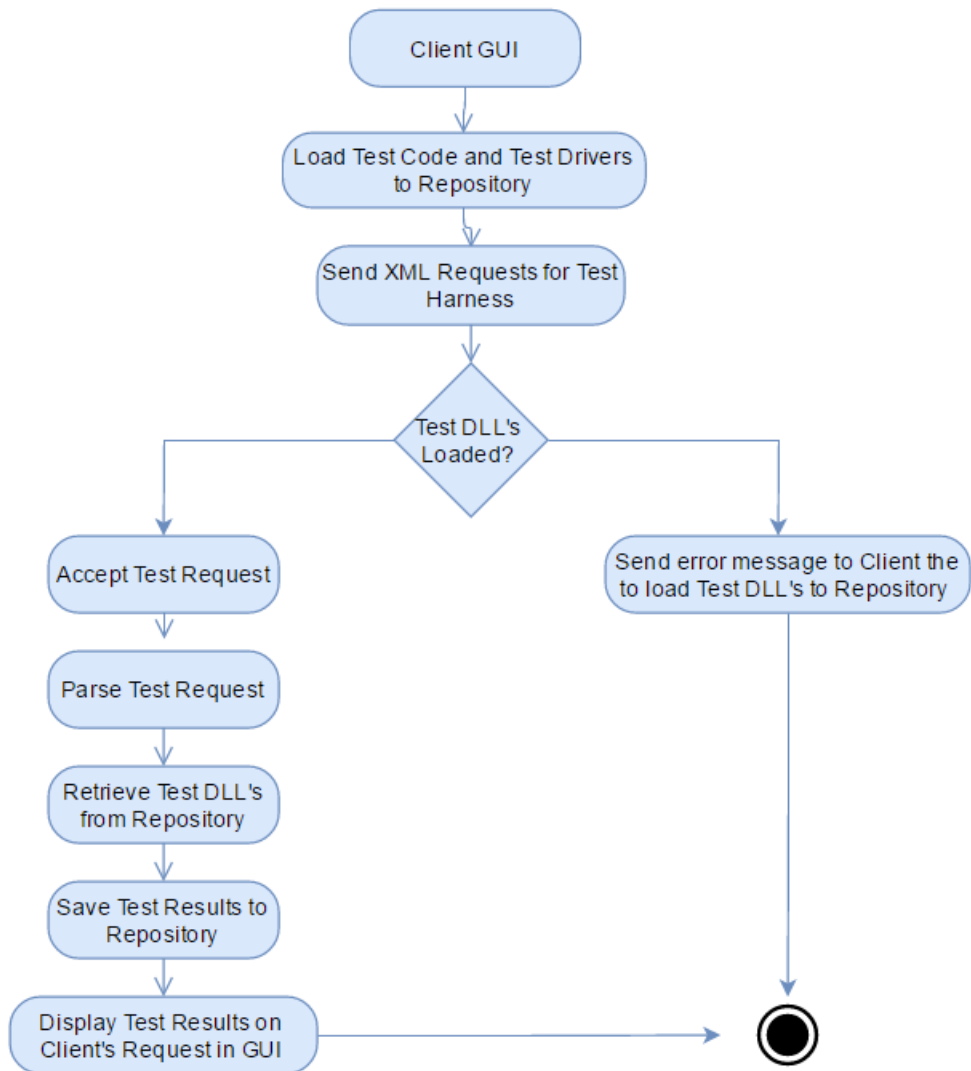


Figure 1 Basic Activities of Test Harness

## 2.2 Obligations

'Test Harness' should be developed in such a manner that it provides ease of use for the user using this tool to support continuous integration tests. It should also offer better performance, security, reliability and flexibility for different user groups. The primary obligations for the system would be:

- Provide mechanism for integration testing.
- Provide an interface for Client to communicate with Test Harness and Repository Server.

- Accept test request in the form of XML file from multiple clients concurrently and queue them sequentially.
- Test Requests should be processed and logged simultaneously for each test request.
- Test requests should be isolated and parsed separately.
- Repository server should contain DLL's for test code and test driver supplied by the client.
- Test Driver runs test and saves the results in the repository as a file for supporting client queries.
- Test results should be logged and displayed on client's request.
- System should include means to time test executions and communication latency.

### 2.3 Organizing Principles

- The primary principle is to develop each package with a specific functionality which will communicate with test harness.
- We use .Net framework AppDomain class for each test request.
- Test results and Test log are stored as Files as a persistent store in a Repository.
- Communication between Repository, Test Harness and Client should be done using WCF and GUI for client is built using WPF.

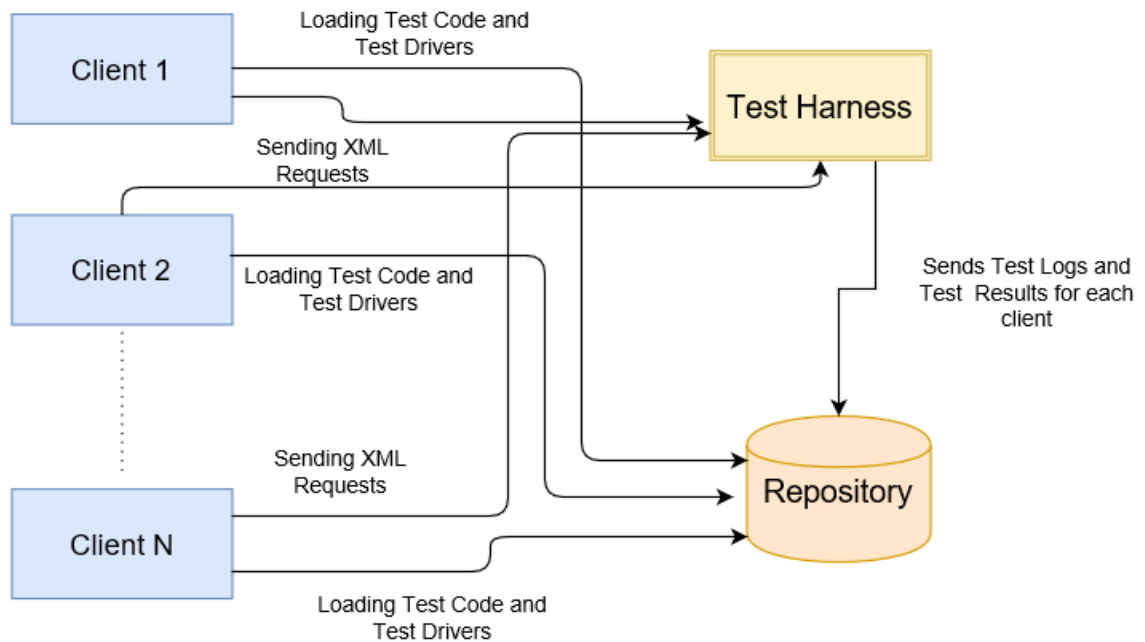
### 2.4 High Level Architecture

The key architectural idea of Test Harness System is depicted in the following figure. There are several clients who want to access the test harness server concurrently. Each client will run on a separate thread in the system. Consider a single client, for each client there exists a separate GUI form which the client can communicate with the Test Harness and Repository Server. First the client loads the test code and test driver which is to be tested in to the repository using GUI provided. The communication between Client, Test Harness and Repository is done using WCF. Then, Repository creates a separate folder for each client with the current date and time and test developer's identity where the respective DLL's for test code and test driver will be saved. The test results after performing the tests will also be saved in the Repository under the same folder.

After loading the DLL's to Repository the client will supply the test requests to Test Harness for performing testing using the same GUI provided for each client. Here client can only send test request to Test Harness after loading the test code and



test drivers in the repository, otherwise test harness will throw an error message saying test code and test drivers are not present in the Repository. Then, test harness pulls the test drivers and all related Dynamic Link Libraries(DLL) from the repository. After that our test harness parses the metadata and runs the test with the help of test drivers. Finally, the test harness stores the test logs and test results in the Repository under the respective client's folder. Then the client can query the test results as per his need from the Repository server. It is to be noted, that in this context that test request in the form of XML (i.e. metadata).



*Figure 2: High Level Architecture of Test Harness System*

The further section of the document will illustrate in detail about the modules and package involved in developing this application.

### 3. Users and Uses

#### 3.1 Actors

Test Harness is an application/tool which runs continuously in the development environment to support continuous integration tests. The following are the users who interact with the Test Harness in various ways.

##### 3.1.1 Project Developers

Developers are the principal users for Test Harness. They are the ones who test the newly developed code with the stable project baseline. Developers use test harness for integration testing. Developers are responsible for supplying the test requests and test code to test harness, which is then responsible for providing the test results to the developer. Developers should supply strong test cases to identify the errors in the newly developed test code accurately. Advantages for having test harness for developers are listed below,

- Decreases the overhead on developers to perform integrating testing at every stage.
- Helps developers to focus on building efficient code instead of performing tests.
- Product implementation problems can be identified by developer and could inform to project manager, team lead or software architect.

##### 3.1.2 Project Managers

Project Managers play a crucial role in development of software product, as they are responsible for delivering quality product within specified time frame to client. Often, while developing a large project several team leads co-ordinate to achieve the functionalities of the project and report the project manager so it becomes easy to track the progress. Following are the ways in which project manager use Test Harness

- Project manager primarily uses test and result log module in test harness system to track the progress of the project.
- Project manager tracks which developer is more active in developing the assigned specifications of the project with the help of test log.
- Project manager can use Logger module to know number of developer's active at particular point of time.
- Access performance of a particular team in the project.

### 3.1.3 Quality Assurance

The main role of the Quality Assurance team is to build confidence in the customer subjected to the final product begin delivered. They emphasis on catching defects and maintaining standards in the final deliverable product. Quality Assurance team will give the test each modules of code before integrating to the project baseline with their own test suits. For this purpose, they use test harness to perform tests on their own test cases.

### 3.1.4 Teaching Assistants and Instructor

Teaching assistants and Instructor will perform tests on the Test Harness system which is going to be built. They will check whether all the requirements of the application are successfully implemented. If there are any faulty construction in the development of Test Harness they will report to the developer. They also check for feasibility of the test harness for future applications.

## 3.2 Uses

Test Harness could be used by applications where testing is automated. It could be considered as a concept or a method which can be customized as required for a particular system. For example, Test Harness could be used for unit tests, integration test, regression tests and so on. Following are some of the applications of test harness could be as follows,

- Agile software development is a methodology where in project which is being build is constantly refactored. It is an iterative and collaborative software development methodology. This kind of development strategy requires testing strategy to attain agility and quality in the software development project. Usually we perform two kind of testing in Agile model i.e. unit testing and integration testing. For this purpose, we can have test harness which automates the testing process.
- In project 4 we are going to implement test harness where multiple users are going to connect to test harness to execute their test requests. This application can be made use for concurrent access.
- Test harness which is developed could be used to test on its own code for validation purpose.
- Test harness could be used for communication protocols to monitor the communication by protocol layer (such as protocol layer, communication layer, data link layer, etc.).
- Test harness could be used to create our own functional tests.

## 4. Partitions

The following section describes the various components involved in building Test Harness and their interaction to achieve functionalities required by our required Test Harness. Below is the package diagram for Test Harness.

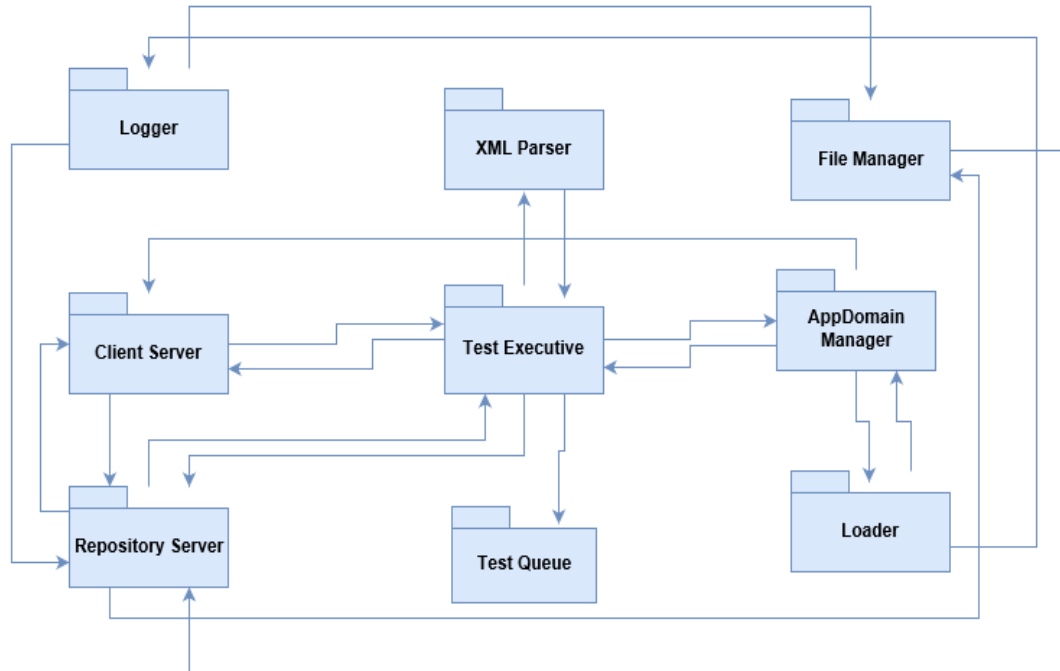
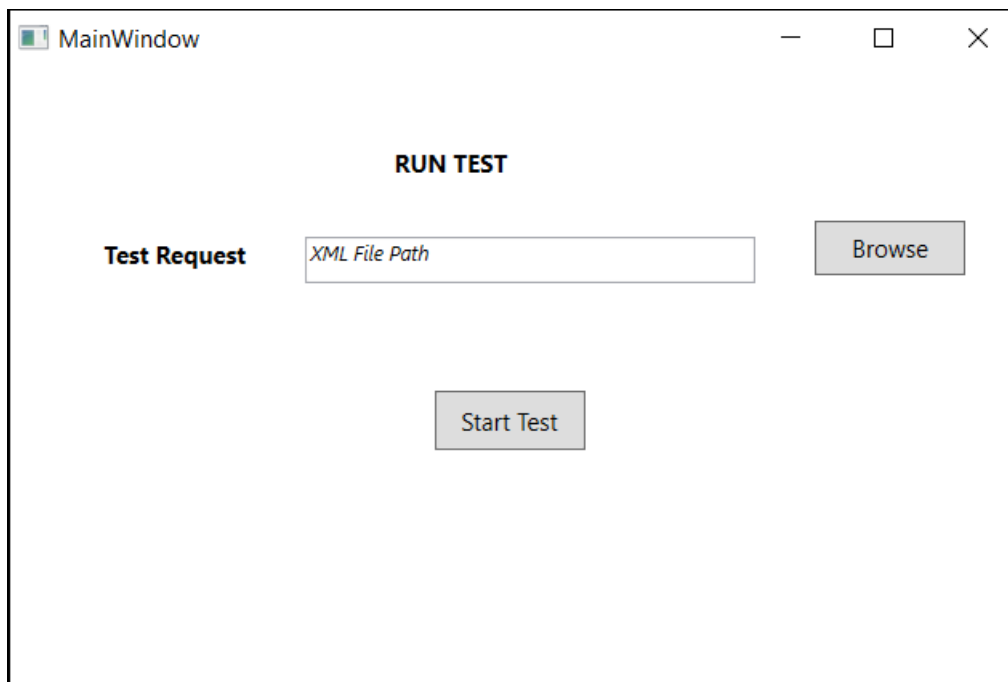
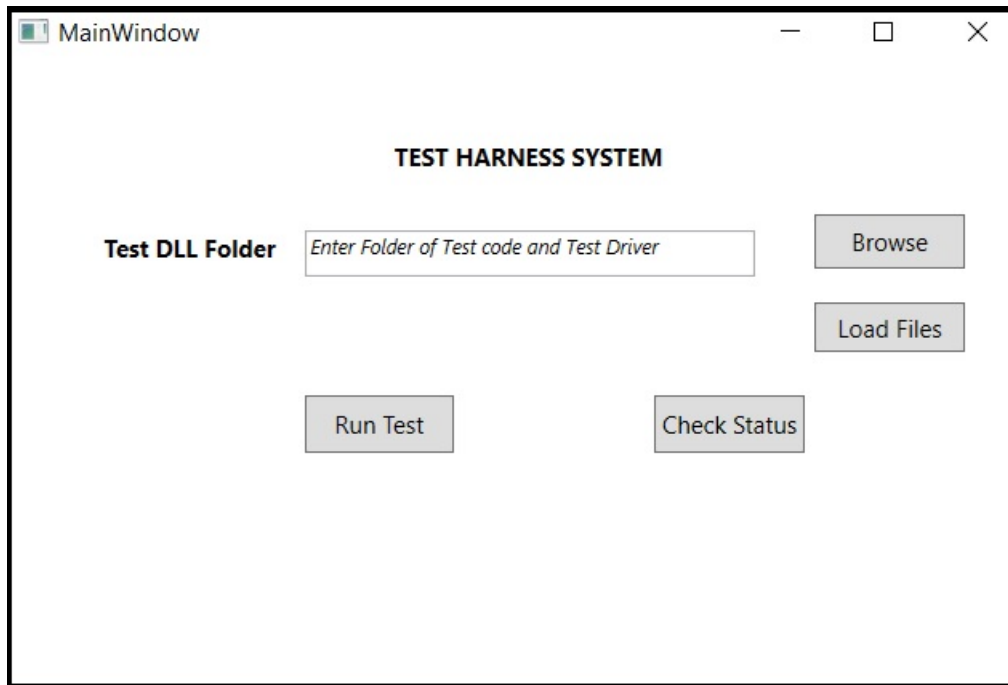


Figure 3 Package Diagram for Test Harness

## 4.1 Client Server

In our system, we have multiple clients simultaneously using Test Harness and Repository Server. Each client is provided with a GUI to communicate with the Test harness and Repository server. The client, repository and test harness are present in different servers. To provide communication between these three we are using Windows Communication Foundation (WCF). The GUI is built with Windows Presentation Foundation (WPF). The sample GUI for the clients is as follows,



Following are the activities performed by the client,

- First the client selects the folder consisting of Test Code and Test Driver which are required to perform tests. The driver and code are in the form of DLL's.

- Then the client loads the files in to the Repository by selecting Load Files button provided in the interface.
- After loading the clients selects Run Test button. Then a new window will pop up for sending the test request for test harness server. Here client sends the XML file as a test request and clicks start test button. By doing so the test harness starts executing.
- Client should run test only after loading the DLL files or else test harness will throw an error message to the client.
- Client can also periodically check the status of the tests using Check Status button.

After the test harness performs the testing the client will receive a new window about the results for their code under testing.

## 4.2 Repository Server

This module contains the actual project baseline and the code to be tested against. It also has a test driver which is to be loaded to execute test cases for the test request of a client. In our implementation, we are building a simple repository which stores the test code and test driver of a client under a separate directory. Following are the responsibilities of the Repository Server.

- For each client, a separate directory should be created with a directory name as client's name and current date time.
- The Repository should also store the Test result data and log under the same folder where a respective client has the DLL's for test code and test driver.
- Repository should provide the Client with the test logs whenever requested.

## 4.3 Test Executive

Test executive controls the main flow of the application. It is the entry point for our application i.e. Test Harness. It takes client's test requests as input in the form of XML and directs the flow to various packages. In our test harness server, each client runs on his own child thread. Interactions of test executive to various packages are as follows,

- Test Executive reads the test request in the form of XML file provided by the client directly, which is considered as inputs to our test harness. Test Harness accepts the requests simultaneously.

- It interacts with Test Queue package to queue the test requests and, XML Parser package to parse the XML to supply the test cases, test drivers and code to the AppDomain Manager Package.
- Test Executive builds a parent AppDomain and for every Test Request running on a thread, it separately builds a child AppDomain. The reason for creating AppDomain is that in C# the only way we can unload DDL is by using AppDomain feature of .Net framework. Also, the reason for creating child AppDomain is that, when an unhandled exception is encountered the entire application doesn't get hung. Test Harness could be given the functionality to display the test results but in real time Test Harness will be always kept busy with request. So the functionality is delegated to AppDomain to store the test log and test results.
- It communicates with Repository server to load the required DLL's
- It associates with Loader package to load the DLL's to Test Harness from the repository server.
- It cooperates with Logger package to store the logs and save it to the repository server.

#### 4.4 Test Queue

This package queues the test requests from various clients. We use a blocking queue for supporting concurrent clients. The blocking queue uses locks to support concurrency because each client runs on a child thread so locking mechanism is a mandate to support multiple clients. Then this information is further processed by Test Executive.

#### 4.5 XML Parser

Test Executive supplies the XML parser with the test requests which must be parsed before sending the parsed data to the Child Appdomain. The parsed data gives information about test drivers, test cases and author to whom the test request belongs. Parser parses the test request sequentially and transmits to Child AppDomain.

#### 4.6 AppDomain

This is the module where actual testing takes places. AppDomain is a way for isolating the running applications. Scalability of the system increases by running

multiple application with in a single process. Exceptions/faults in one AppDomain doesn't affect application running in other AppDomain. In our application Test executive creates a parent AppDomain and for every test request running on a child thread a separate child app domain is created. Responsibilities of AppDomain are as follows

- Create separate child domain for each test request.
- Execute the test cases by loading the test drivers for the test request form the repository.
- Save the test log information and test results.
- The saved results are sent to repository where client can query his test results.

#### 4.7 Loader

This module is responsible for loading the DLL in to the AppDomain for executing the tests. It loads test driver along with test code to AppDomain. Here we could perform tests after loading the tests. The Test is performed by using the interface which implements ITest interface for each driver. The interface mainly consists of two important functions test() which returns a status of test i.e. pass or fail and getLog() which returns the log in string format.

#### 4.8 Test Logger

This package handles all the functionalities of logging. Test Harness Application stores test results and logs for each of the test executions using a key that combines the test developer identity and the current date-time. The test results are store in database so that if required project manager, developer or quality assurance could analyze the data for prospects.

#### 4.9 File Manager

File Manager consists of file managing utilities such as searching for DLL's form repository to supply to the loader, creating file for saving the test logs, creating directories in the repository for each client.



#### 4.10 Optional Modules

Apart from the mentioned modules we could also incorporate some modules which are mentioned as follows

- Version control: Every time when a user updates a project baseline it becomes necessary to update the version information and notified to all the users. If version information is not maintained each developer may create code for different versions and may have unexpected errors while adding the new code to the project baseline. In our implementation of project 2 we are not considering multiple users so this module need not be added.
- Authentication and Authorization: To improve the security of the system including this module would be beneficial. In our Test harness application, we have three major users viz. Developers, Project Managers and Quality Assurance and each will have different privileges to access test harness. Having this module will increase the security of the application.

## 5. Activity Diagram

Activity diagram shows the sequence of activities, behavior or workflow of the business system. Below is the activity diagram of the whole system.

### 5.1 Activity Diagram for Loading DLL's in to the Repository

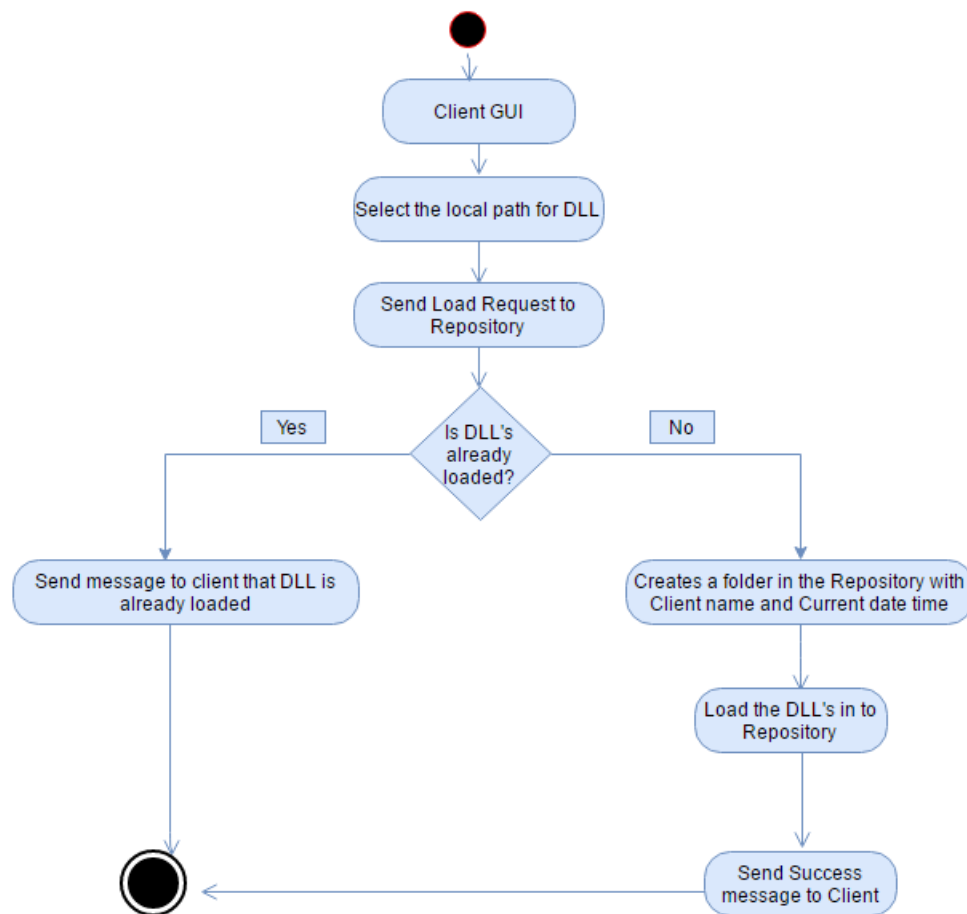


Figure 4: Activity Diagram for Loading DLL's in to the Repository

**Description:** The activity diagram in figure 4 shows the various functionalities and activities performed to load the DLL's of the Test code and the Test Driver in to the repository. The activity diagram is for a single client but we can say multiple clients access the repository simultaneously.

- First every client is provided with a GUI from which he could load the DLL's in to the Repository specifying the path of the local directory where the DLLs are present.
- Then client will initiate the load request to the repository server. The repository server accepts the load request from multiple clients and processes simultaneously.
- If the client has already loaded the test request in to the Repository, then the repository server will send a message to the client stating that the DLL's for test code and test driver are already loaded.
- If the DLL's are not loaded, then the Repository will create a folder for a client with client's name and current date time.
- After creating the directory, the test code and test driver are placed in that repository.
- Finally, a success message is passed to the client by the repository server.

## 5.2 Activity Diagram for Testing the Code and Displaying the Results

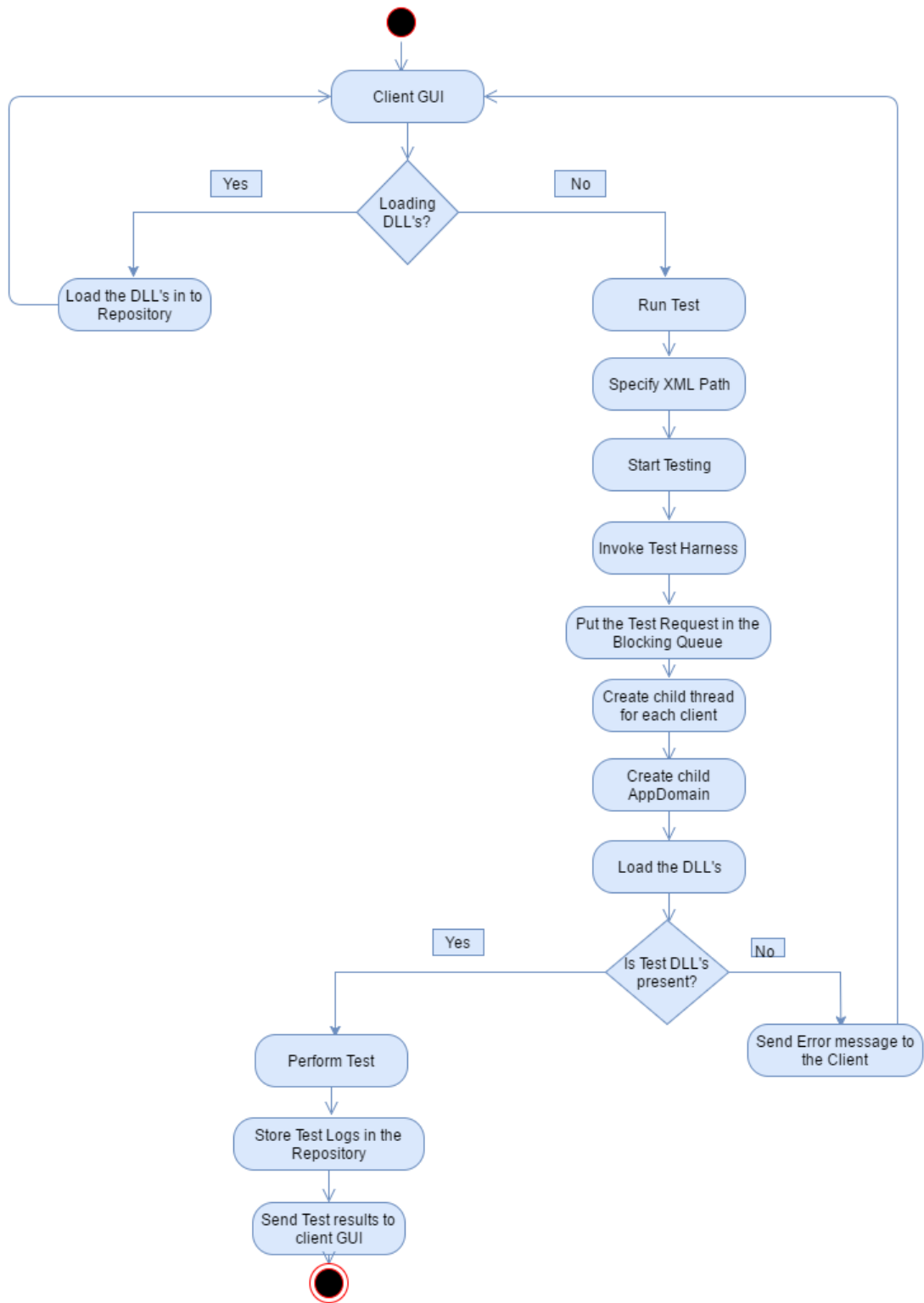


Figure 5: Activity Diagram for Testing the Code and Displaying the Results

**Description:** The activity diagram in the Figure 5 represents the interaction of client with the test harness to perform the testing. The activity diagram is for the single client but it could be extended for multiple clients accessing test harness simultaneously. Following are the activities performed,

- Client is provided with the GUI to run the tests. When the client selects the run, test option the GUI will prompt for supplying the test request in the form of XML file by specifying the path of the XML file in the local client.
- Once the client selects the start option, the test harness gets invoked and further processing takes place.
- Once the test harness receives the test request it is put into blocking queue which consist of locks to support multiple user requests.
- Then the test harness creates a child thread for each client on which test request runs on a separate child AppDomain.
- Then the child AppDomain loads the test code and test driver DLLs from the repository server and performs the test. If the DLLs are not present, then test harness will raise an error message to the client to load the DLL.
- Once the test is performed the child AppDomain send the test results to the Client GUI. Also, the test results and logs are stored in the repository server to support client queries.

## 6. Critical Issues and Analysis

This section will discuss various critical issues which has to be considered while designing Test Harness application. If the critical issues are no addressed it may lead to flaw in the system which is being developed. Some critical issues and their potential solutions are mentioned below.

### I. Performance of Test Harness for Large Number of Test Requests from multiple Clients

In general, Test Harness is going to be the busiest server in the software development because it continuously accepts test request from multiple clients and processes the tests and provide the results. So, performance consideration is a critical issue in the development of test harness system.

**Solution:** To solve this problem we might introduce threading, which means each client will run on a separate thread for their test requests. Which makes number of clients accessing the test harness concurrently. Also, to improve the performance of the test harness we are introducing the child AppDomain so that any unhandled exception won't cause failure in the entire test harness system.

### II. Ease of use for the Users

The system which is being constructed must be easy to use for the clients. If the system which is being constructed is not easy, the client might not use the test harness for testing. So, the interface provided for the client should be easy to understand use accurately.

**Solution:** We are providing a Graphical User Interface for all the clients for the system which helps them communicate with the repository and test harness server easily. We will also provide GUI to support client queries about the test log and results which is generated after performing the test.

### III. Impact of XML Parsing on Test Harness.

In our implementation, we are parsing XML test request and then passing the parsed input to the child AppDomain for performing tests. Suppose if the client provides the XML test request other than the specified schema for the test request the test harness should be handle such scenarios which makes this a critical issue.

**Solution:** We should provide specified schema for the XML files for the clients to use for supplying the test requests to test harness. Which makes the parsing of test request by the test harness becomes easier. If the client does not use the format test harness will raise an error message to the client.

#### IV. Missing DLL's in the Repository

If the client supplies the test requests in the form of XML files to the test harness server before loading the DLL's of the test code and test driver, then the test harness cannot perform the test because of the missing DLL's in the Repository.

**Solution:** When the client runs the test request first without loading the DLL's we are going to prompt the respective client with the error message stating that DLL's for the test code and test driver mentioned in the test request are not present. Thus, we handle this critical issue.

#### V. Version Control

Test Harness tests the newly developed code with the project baseline and returns the test results. Sometime the baseline code may occur changes because of newly tested functional module being added to the project baseline. In this case all the users utilizing test harness will need to know that project baseline code changed to newer version so that each user can build code and configure test cases according to the newer version of project.

**Solution:** The baseline repository needs a version control module associated to it which tells the current version of the project baseline to the users along with information regarding previous versions of the baseline. In our implementation, we are just building a simple repository which has basic functionalities. Version control is not implemented in our Repository as it is not a requirement.

#### VI. Authentication and Authorization

In a development environment where a large system is being built there will be several users involved like Developers, Project Managers, Quality Assurance and Testers. To improve the security of system each user will have their own privileges and should be authentic to log into the system. Every user does not require all the functionalities of the Test Harness.

**Solution:** Adding an Authentication and Authorization module in a multiple client environment will increase security but we don't have to implement this module for demonstration.

#### VII. Handling Exceptions for Test Requests

In some scenarios input files, i.e. XML may have some missing data which may hang the entire system if it is not managed properly. Also while executing the

test some rare exceptions may arise which leads to undefined state and eventually interfere with normal functioning of Test Harness or probably failure of the entire system.

**Solution:** For this purpose, we may have an Exception Handling module which interacts with the test executive to handle exception occurring at any point. We have isolated test request execution by using child AppDomain so that unhandled exception doesn't bring down the entire system.

#### VIII. Demonstrating Requirements

We need to show that our project meets the specification stated by the instructor to Teaching Assistants which is one of the critical issue.

**Solution:** Test Executive package takes care of automating the process of testing specifications and then displays the results in the output console.

## 7. Conclusion

To summarize, Test harness is a software tool which is generally used in development of large applications to automate the process of integration testing. It reduces the burden on Developers to focus on integration testing and concentrate more on the functionalities of new code being developed. We have discussed various actor viz. developers, quality assurance and project managers and their uses for test harness. This OCD consists of package diagrams which represent the different modules and their interactions.

This document also describes the activity diagram which describes work flow and set of activities for Test Harness. Finally, we mentioned few critical issues along with solution which help in developing a stable and reliable system with good design.



## 8. Appendix

### 8.1 Message Creation and Parsing Package

The following is the code prototype for message creation and parsing. The code specifies the structure of the message which is used between client and test Harness Server. This prototypes shows how the message is created and transmitted in the WCF communication between client and server.

```
[ServiceContract]
public interface ICommunication
{

    [OperationContract(IsOneWay=true)]
    void PostMessage(Message msg);

    // Not a service operation so only server can call

    Message GetMessage();
}

[DataContract]
public class Message
{
    [DataMember]
    Command cmd = Command.XMLTestRequest;
    [DataMember]
    string body = "default message text";

    public enum Command
    {
        [EnumMember]
        XMLTestRequest,
        [EnumMember]
        Status,
        [EnumMember]
        DoAnother
    }

    [DataMember]
    public Command command
    {

```

```

        get { return cmd; }
        set { cmd = value; }
    }

    [DataMember]
    public string text
    {
        get { return body; }
        set { body = value; }
    }
}
}

```

## 8.2 Message passing communication package(s) that send messages between local clients and a remote server.

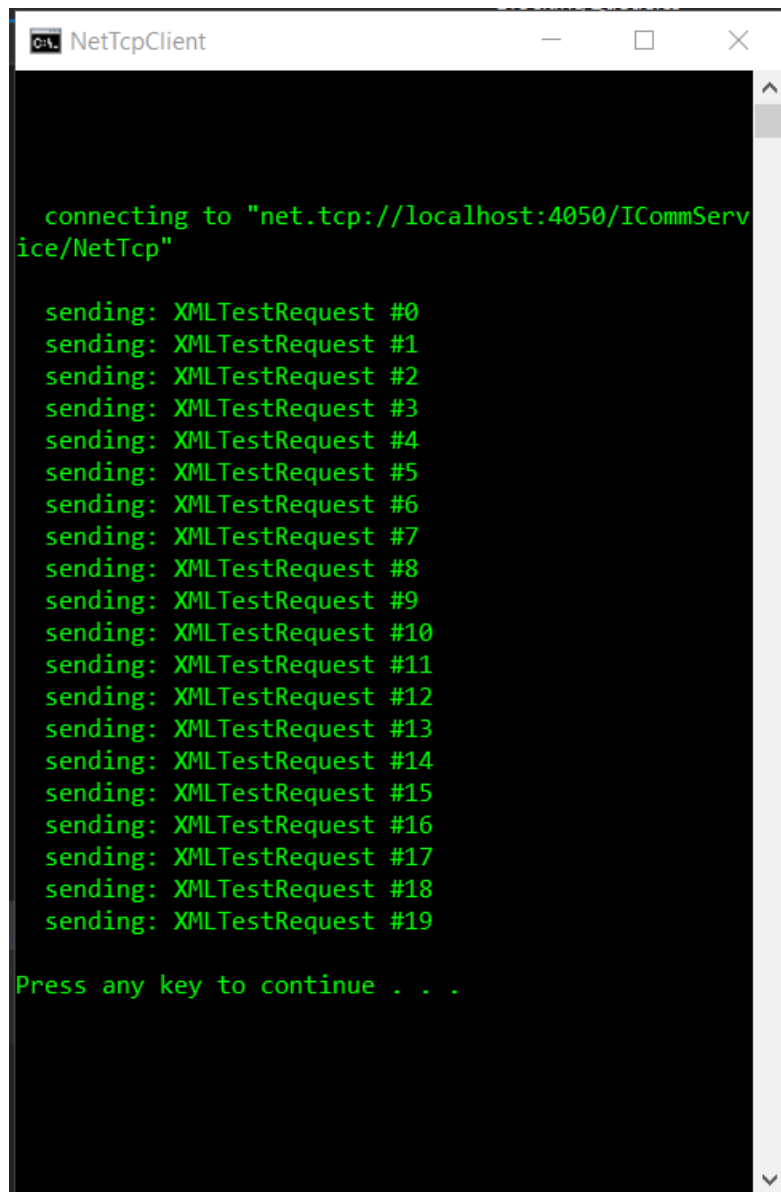
The following prototypes are for Message passing communication package that send message between local server and remote client. Here we are using NetTcp binding provided by WCF to create a communication channel so that message passing can occur between our local client and remote Test Harness server. It primarily has three files, CommService.svc.cs, ICommunication interface and NetTcpClient.cs.

Following output show the communication between client server and test harness. The client is sending the xml requests to the test harness and the test harness is receiving the xml test requests from the client which show that a communication is established between the two.

```
Test Harness
Test Harness Starting up
=====

Test Harness is ready.
Maximum NetTcp message size      = 65536

received: XML Test Request      XMLTestRequest #0
received: XML Test Request      XMLTestRequest #1
received: XML Test Request      XMLTestRequest #2
received: XML Test Request      XMLTestRequest #3
received: XML Test Request      XMLTestRequest #4
received: XML Test Request      XMLTestRequest #5
received: XML Test Request      XMLTestRequest #6
received: XML Test Request      XMLTestRequest #7
received: XML Test Request      XMLTestRequest #8
received: XML Test Request      XMLTestRequest #9
received: XML Test Request      XMLTestRequest #10
received: XML Test Request      XMLTestRequest #11
received: XML Test Request      XMLTestRequest #12
received: XML Test Request      XMLTestRequest #13
received: XML Test Request      XMLTestRequest #14
received: XML Test Request      XMLTestRequest #15
received: XML Test Request      XMLTestRequest #16
received: XML Test Request      XMLTestRequest #17
received: XML Test Request      XMLTestRequest #18
received: XML Test Request      XMLTestRequest #19
```



```
NetTcpClient

connecting to "net.tcp://localhost:4050/ICommService/NetTcp"

sending: XMLTestRequest #0
sending: XMLTestRequest #1
sending: XMLTestRequest #2
sending: XMLTestRequest #3
sending: XMLTestRequest #4
sending: XMLTestRequest #5
sending: XMLTestRequest #6
sending: XMLTestRequest #7
sending: XMLTestRequest #8
sending: XMLTestRequest #9
sending: XMLTestRequest #10
sending: XMLTestRequest #11
sending: XMLTestRequest #12
sending: XMLTestRequest #13
sending: XMLTestRequest #14
sending: XMLTestRequest #15
sending: XMLTestRequest #16
sending: XMLTestRequest #17
sending: XMLTestRequest #18
sending: XMLTestRequest #19

Press any key to continue . . .
```

## 9. References

1. [https://msdn.microsoft.com/en-us/library/ms173138\(v=vs.90\).aspx](https://msdn.microsoft.com/en-us/library/ms173138(v=vs.90).aspx)
2. <http://www.ecs.syr.edu/fawcett/handouts/Webpages/BlogTesting.htm>
3. <http://www.ecs.syr.edu/fawcett/handouts/CSE681/lectures/Project2-F2016.htm>
4. [https://en.wikipedia.org/wiki/Test\\_harness](https://en.wikipedia.org/wiki/Test_harness)
5. Project starter code given by Prof. Jim Fawcett