

CSE 681- Software Modeling and Analysis

# Test Harness

Project 1 - OCD

Moneesha Modi (SU ID : 861810128)  
9/14/2016

## Table of Contents

1.	Execution Summary .....	3
2.	Introduction .....	5
2.1	Application Obligations.....	5
2.2	Organizing Principles.....	5
2.3	Key Architectural Ideas .....	6
3.	Users and Uses.....	7
3.1	Developers .....	7
3.2	Instructor/Teaching Assistant/Graders.....	7
3.4	Quality Assurance (QA) .....	7
3.5	Extending to Project 4.....	8
4.	Partitions.....	9
4.1	Test Executive .....	9
4.2	Demo Client .....	10
4.3	Blocking Queue .....	10
4.4	XML Parser .....	10
4.5	Repository .....	11
4.6	File Manager .....	11
4.8	Application Domain Manager .....	11
4.9	Logging Manager.....	12
4.10	Database .....	13
5.	Application Activities .....	14
5.1	High Level Activity Diagram .....	14
5.2	Application Domain Manager Activity Diagram.....	16
6.	Critical Issues.....	18
6.1	Ease of Use.....	18
6.2	Performance and Productivity .....	18
6.5	Logging .....	19
6.6	Security .....	20
6.7	Extending to Project 4.....	20
7.	References .....	21

## 1. Execution Summary

In large software development companies, the software developed consists of millions of lines of code and is developed by hundreds of developers. Every developer perhaps is responsible for a set of modules which is developed, tested and then needs to be integrated. This integration needs you to combine the modules and packages developed by the developers after individually testing each module. Testing each package or module is an extremely tedious and time consuming process especially for large software systems. Hence comes into scenario the concept of a test harness. A test harness interacts with the client for input of test code and automates the process of testing multiple packages. It sends the test results to the user through messaging. The tested code is then added to the baseline code. The test harness can be used to perform various forms of tests such as Construction Tests, Unit Tests, Regression Tests, Integration Tests, Qualification Tests and Performance Tests.

The key features of this system can be illustrated as follows:

- **Test Executive:** This module will demonstrate the user the working of the entire system and ensure that all the requirements are met.
- **Application Domain Manager:** This module will enable the test harness to isolate each of test requests consisting of test codes to run separately and records the test results.
- **Loader:** This package will assist the application domain manager to load the dynamic link libraries associated with each test driver and its test code.
- **Logger:** The logging manager will record the details of all the testing performed such as timestamp, test cases tested, results of the tested test cases etc.
- **Messaging:** The test driver and test code to be tested will be specified to the test harness through this module via XML files. The test results will also be communicated to the user via this feature.
- **Demo Client:** This package will feature a command line interface for the user to provide the code to be tested.

The primary users for this system are the **developers, instructors, teaching assistants, QAs** and the **managers**. A few critical issues shall be considered while developing the system and their solutions would be provided. Some of these are:

- **Ease of Use:**

- Performance
- Demonstration
- Accuracy
- Security
- Logging
- Extending to Project 4

The test harness will therefore be excellent for automated testing and will reduce the load of the users to a great extent. This document shall explain all the functions of the system, users at various levels, application activities, interactions between various packages, modules and the structure of the test harness in detail. It will also provide discussion on the various critical issues and the solutions recommended for these issues.

## 2. Introduction

In the recent age of technology and communication, Information Technology plays a huge role in all aspects of our lives. There are millions of people seeking jobs in this industry and building companies on this front. There are huge IT companies such as Microsoft, Google, Amazon, Apple, VMWare, Yahoo, Facebook and IBM where billions of lines of code is being developed everyday by millions of software developers. This code needs to be tested again and again by developers, testers and managers. To make the lives easier of these IT professionals easy, we need to develop a test harness to automate the test procedure and monitor the results. The test harness should enable the tests to run multiple times or at fixed intervals. The test harness should provide the following benefits:

- High productivity and performance
- Improved quality of software being delivered
- Reduce manual labor and intervention

### 2.1 Application Obligations

The key feature of the test harness is to automate the testing process such that the procedure becomes less complicated, easy to use, fast and robust. The primary obligations of the system are:

- Creating a repository for all the dependent code, test drivers and tested code
- Sending out test requests to test harness through XML files
- Parse the XML files and queue the requests with the help of a blocking queue.
- Using application domain managers to isolate each of the test requests so that they run separately on the child app domain.
- Creating dynamic link libraries (dlls) through C# code for each of the tests that need to be run.
- Using a loader to load the dynamic link libraries into the application domain managers

### 1.1 Organizing Principles

The organizing principles of the system are to perform the primary functions of the test harness and provide solutions to all the critical issues. The system should make

use of various C# factory methods, test drivers; different packages for each features, dynamic link libraries and application domain managers and XML files.

## **1.2 Key Architectural Ideas**

In our system, we will create a directory consisting of all the baseline code, test drivers and the test code in the form of dynamic link libraries. This directory serves as our repository. This system will be built in C# using the .Net framework and Visual Studio 2015. The users will interact with the test harness through a command line interface and send out test requests in the form of XML files. These test requests are enqueued in a blocking queue and dequeued one at a time. Every request is parsed and the required dynamic link libraries are fetched from the repository. The dynamic link library for every test request is run on a separate application domain with the help of a loader. Finally the test results are sent to the user and the results are logged in the database.

## **2. Users and Uses**

### **2.1 Developers**

The developers can use the test harness to automate their testing process and generate test results of the code they have developed. The developers can specify the names of the test drivers and the test code libraries through an XML file. These libraries will be fetched from the repository and run on different application domains to get a “Pass” or “Fail” test result for every test case in the test driver.

### **2.2 Instructor/Teaching Assistant/Graders**

The instructor, teaching assistant and graders can use this tool to examine the system and check if all the requirements are met or not. They will be provided with a test executive by the developers which will demonstrate all the requirements of the test harness.

### **2.3 Managers**

The managers in a software company can use the test harness to check the log messages which are recorded by the logging manager. The log messages can provide them the information of all the code that has been tested in a current period of time and their corresponding results i.e. whether a particular test case has passed or failed. By this process, the managers can constantly monitor the progress made by the developers and can track their activities.

### **2.4 Quality Assurance (QA)**

The quality assurance team in various software companies perform testing before the code is delivered or deployed. Hence they can also use the test harness to perform qualification tests, performance tests and regression tests. They can use the test results to determine if the code is fit to be deployed or not.

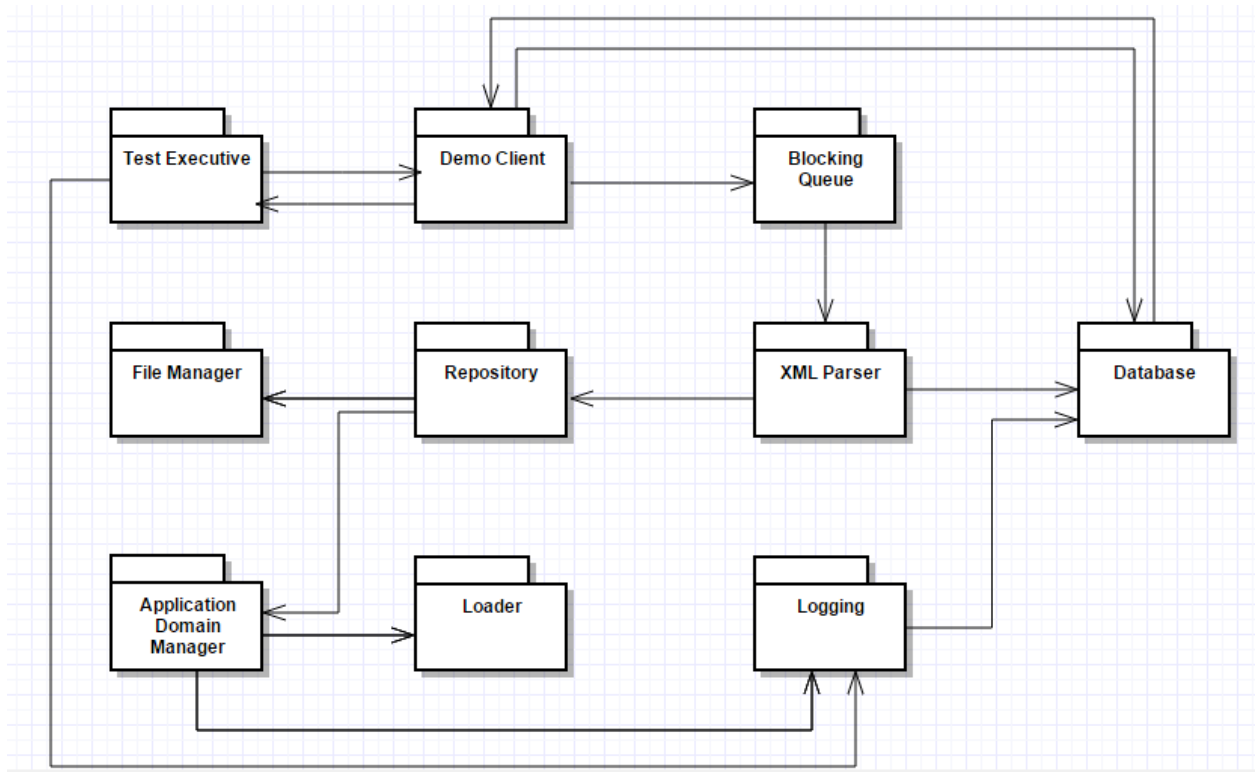
## **2.5 Extending to Project 4**

In Project 4, we will be converting the test harness to run on remote machines on different threads thereby increasing the impact of the system. This will enable the test harness to run simultaneously on various machines and test different test drivers on different machines. It will improve the speed of the entire testing process. It also enables the system to be scalable, robust and perform better.



### 3. Partitions

The test harness is divided into various packages which perform a set of functions and interact among each other. The package diagram for our system is illustrated below:



The basic functionalities of all the packages are described below:

#### 3.1 Test Executive

The test executive is the main entry point of the application. This package provides a batch file to the user which will demonstrate that all the requirements of the system have been met and that the system is working as expected. The test executive shall demonstrate a few test cases which are successful, a few test cases that fail. It will also test a part of the code of the test harness.

The main method resides in this package and shall make a call to the Demo Client package which will provide a command line interface to the user.

The test executive can make calls to the following other packages to perform other functions:

- i) Demo Client to show the test results on the console
- ii) Logging to display log messages to the manager
- iii) Database to retrieve test results and test logs.

### **3.2 Demo Client**

The demo client package displays a command line interface to the user when called by the Test executive package. The demo client parses the command line arguments provided in the test harness. These command line arguments will be a path of the repository, type of user and a directory which consists of the XML test requests that need to be run on the test harness. These test requests are retrieved from the specified directory and sent to the Blocking Queue package.

### **3.3 Blocking Queue**

The Blocking Queue package provides a mechanism for the XML files to be tested one at a time. It consists of two operations:

#### **3.3.1 Enqueue**

The enqueue operation adds all the XML files to the queue when the test harness calls the Blocking Queue package.

#### **3.3.2 Dequeue**

Once all the XML files are enqueued in the blocking queue, one XML file is dequeued at a time. If the queue is empty when the reader attempts to dequeue an item then the reader will block until the writing thread enqueues an item. The blocking queue is implemented using a monitor and lock to make the waiting efficient. The dequeue operation calls the XML parser package for decoding the XML files

### **3.4 XML Parser**

When the XML parser receives the XML document, from blocking queue, the document is parsed by the XDocument class which is represented by a XML document under the System.Xml.Linq namespace in the C# library. The parser retrieves the names of the test drivers and libraries from the XML file and stores them in a List. After the document has been parsed, the Repository package is called. A sample XML file has been included in the appendix.

### **3.5 Repository**

The Repository consists of a directory which consists of all the dynamic link libraries of the dependent code, test drivers and the code to be tested. When the XML Parser makes a call to the Repository package, the repository scans through the list of libraries to be tested and checks which are the dependent libraries for the libraries mentioned in the XML input. The dependent libraries are also added to the list of required libraries. The repository also consists of the path at which it resides on the machine. The repository calls the file manager package and sends the list of libraries to the file manager.

### **3.6 File Manager**

The File Manager package accepts the list of libraries required to test the code from the repository and runs a scan from the directory where the repository is placed. It searches for all the \*.dll files, compares them with the required list of libraries and then extracts the path of each of the libraries to be tested.

### **3.7 Loader**

The loader is responsible for loading the dynamic link libraries for each of the test drivers and their test codes in individual application domain managers so that they are isolated and run separately from each other.

### **3.8 Application Domain Manager**

Each .NET process usually hosts just one application domain: the default domain, created automatically by the CLR when the process starts. It's also possible and

sometimes useful to create additional application domains within the same process. This provides isolation while avoiding the overhead and communication complications that arise with having separate processes. It's useful in scenarios such as load testing and application patching, and in implementing robust error-recovery mechanisms.

### 3.8.1 Creating an Application Domain

```
// Create application domain setup information for new AppDomain

AppDomainSetup domaininfo = new AppDomainSetup();
domaininfo.ApplicationBase = "file:///" + System.Environment.CurrentDirectory;

// defines search path for assemblies
//Create evidence for the new AppDomain from evidence of current

Evidence adevidence = AppDomain.CurrentDomain.Evidence;

// Create Child AppDomain

AppDomain ad = AppDomain.CreateDomain("ChildDomain", adevidence,
domaininfo);
```

### 3.8.2 Unloading an Application Domain

```
// unloading ChildDomain, and so unloading the library
AppDomain.Unload(ad);
```

The application domain manager is responsible for isolating each of the test requests so that they run separately on child app domains. The loader is injected in the primary application domain. The dynamic link libraries are loaded by the loader in their individual app domains. Once the application domains have run the code, the results of each test case is inserted in the database and sent to the demo client for display to the user using the `getLog()` function of the `ITest` interface. This function returns a string consisting of the test results of the request. The logs of the test case are sent to the Logging Manager package which are further stored in the database and can be accessed by the manager whenever required.

## 3.9 Logging Manager

The logging manager keeps a track of all the details of the test harness. It constantly monitors the following attributes of the system:

- i) Timestamp of when the test harness was last run
- ii) Test cases that were executed in the past 24 hours
- iii) Test cases that were executed in the past one week
- iv) Test cases that were executed in the past one month
- v) Test results of the test cases executed in the past 24 hours
- vi) Test results of the test cases executed in the past one week
- vii) Test results of the test cases executed in the past one month
- viii) Modules tested in every run
- ix) Increase of modules in the baseline code in the past one week
- x) Increase of modules in the baseline code in the past one month

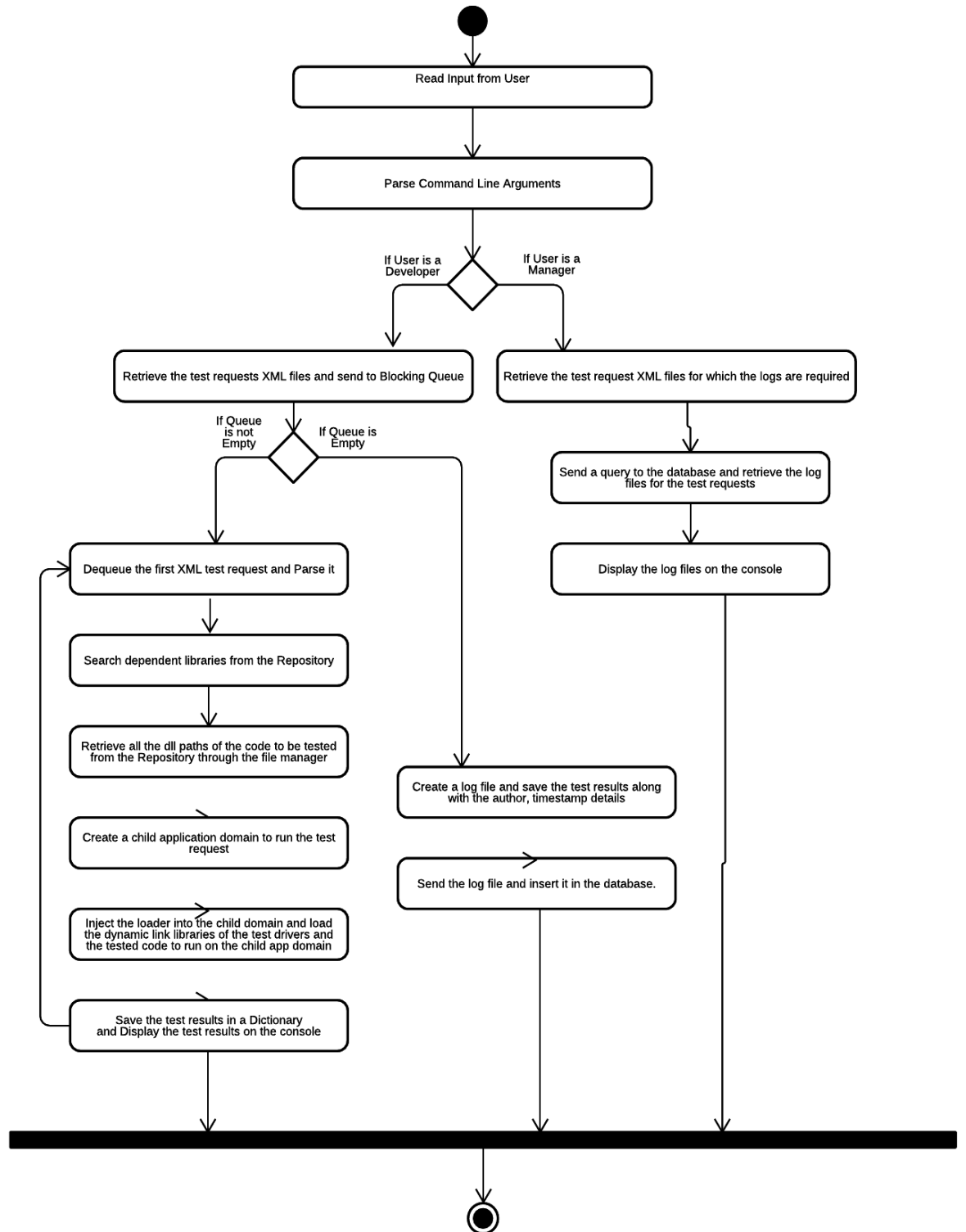
The above attributes will be sent to the database through query processing using a feature called ADO.Net in C#

### **3.10 Database**

We will be using the NoSQL database MongoDB for storing the test results and the test result logs. When a user wants details regarding a specific test run, we can query the database and retrieve the results and send it to the Demo Client which will display the log details of that test run on the console for the user to see.

## 4. Application Activities

### 4.1 High Level Activity Diagram



The above activity diagram describes the flow of activities which will occur in the test harness that we will build. Following are the steps of action for the entire system:

- 1) When the test executive runs, a few command line arguments are passed to the system. These are:
  - a) Path of the repository
  - b) Type of User
  - c) Path of test requests
- 2) The type of the user is checked. If the user is a manager, he just needs to access log files of previous test requests while if the user is a developer, he will actually perform test automations on the test harness.
- 3) If the type of user is a 'Developer', then the xml test requests are retrieved from the path specified in the command line arguments and these files are enqueued in the blocking queue.
- 4) If the queue is empty, then it just waits until a request is enqueued in the blocking queue and updates the database with previous log files, if any.
- 5) If the queue is not empty, then the first test request from the blocking queue is dequeued and passed to the XML parser.
- 6) The XML Parser decodes the dynamic link libraries of the test drivers and the associated code to test from this XML file and adds it to a list. The list is sent to the Repository
- 7) The repository is a directory consisting of the baseline code which is at a path specified as a command line argument, accepts the list from the parser, checks if the code to be tested is dependent on any libraries from the baseline code and adds the dependent libraries to the list of libraries.
- 8) The file manager accesses the repository and retrieves the paths of the dynamic link libraries required based on the list of libraries
- 9) This list is then sent to the Application domain manager which creates a child domain to run the test request.
- 10) The loader is injected in the application domain manager and the dynamic link libraries are loaded in it. The test requests run separately on different child app domains
- 11) Once the application domain manager starts running the test requests, the logger manager is called to log the results of each test driver and its test cases.
- 12) The test results for each test request are saved in a directory and then sent to demo client so that they are displayed to the user.
- 13) After this the control goes back to the blocking queue and checks if there are any more test requests enqueued and if there are, then the next test request is dequeued

and the entire process from step 5 to 13 starts and executes all over again till the blocking queue becomes empty.

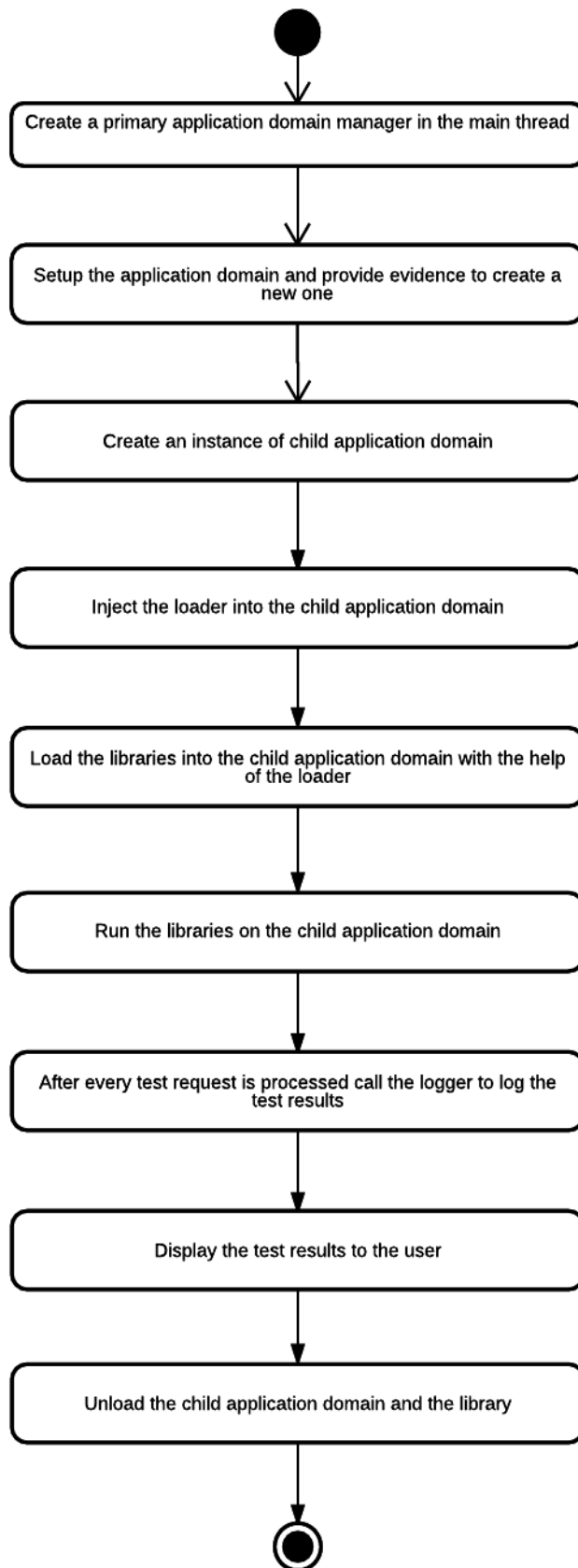
- 14) Once all the test requests are processed, then we call the getLog() method to get the logging details such as the author, timestamp, test results, test cases and the entire report of all the test requests.
- 15) We can create a log file and insert the data to a database by establishing a connection using ADO.Net in C#.
- 16) If the type of user is a 'Manager' then we retrieve the list of XML test request files and send it to the logger.
- 17) The logger checks the database for the latest test run of these libraries through query processing.
- 18) The database returns the results through a log file.
- 19) The contents of this log file are displayed to the user on the console.

#### **4.2 Application Domain Manager Activity Diagram**

The detailed functioning of the application domain manager can be given through the following steps:

- 1) The primary application domain manager is created for the test harness to run.
- 2) The app domain is setup by setting up parameters of the application domain such as Application Base and Evidence.
- 3) When the test requests are ready to execute and all the dynamic link libraries associated with the test request have been retrieved, then the primary app domain is called to create a child domain to run the test request.
- 4) After the instance of the child app domain is created, the loader is injected into the app domain.
- 5) The libraries associated with the test code and the dependent libraries are loaded into the child domain using the ITest Interface.
- 6) The libraries are run on the child application domain using the test() method of the ITest interface.
- 7) After the test request is executed, the logger is called to log the details of the test request and test results.
- 8) The test results are displayed to the user on the console after every test request is executed.
- 9) The test results are stored in a dictionary with a key value pair where the key is the test request and the value is the result in the form of "Pass" or "Fail".
- 10) The ITest interface also consists of a getLog() method which gets a string consisting of all the details of the test request. This is inserted into the database.





## **5. Critical Issues**

### **5.1 Ease of Use**

The first and foremost critical issue of a test harness is the simplicity and user friendliness of the system. The main purpose of a test harness is to make life easy for the IT professionalism. The test harness has a lot of scope to this since the entire process of testing can be automated and they can leave the test harness for hours with a queue of test requests without any manual intervention. But this can be achieved only if the system is simple and easy to use.

#### **Solution:**

The solution to this issue is to create a user friendly interface for the test harness which does not ask for too many input parameters. Hence, we can pass the path of the repository and test request along with the type of user. We can also address this issue further by construction of a graphical user interface in the further project 4 where user can choose the type of user from a combo box and can browse through the file system to choose the path of the test request and the repository.

### **5.2 Performance and Productivity**

The performance of the test harness refers to the complexity of the algorithms used to design the system. We can measure this by the time complexity and the space complexity of the system. The productivity of the system depends on the usage and the comparison of the time taken to perform the test requests through the test harness and manually.

#### **Solution:**

To achieve low time complexity, we must make sure that the interaction between various packages occurs in an efficient manner without any deadlock conditions. Also, we can monitor the number of nested loops. The space complexity can also be reduced by reducing the number of data structures such as arrays, lists and generics such as dictionaries, hashtables etc used for the development of the system.

### **5.3 Demonstration**

Demonstration of the test harness is an important factor when we consider the flexibility of the system. In order to project the proper working of the system, the complete set of requirements need to be showcased to the user so that the user understands the working of the system in an adequate manner. Also, all the functionalities of the system need to be properly tested.

**Solution:**

The solution to demonstrating requirements can be fulfilled through the test executive package which demonstrates the working of the test harness by testing its requirements through the following test requests as a developer type of user:

- i) Test code that generates test result "Pass"
- ii) Test code that generates test result "Fail"
- iii) Test code that tests the test harness code

The creation of log files and insertion of the database can be demonstrated by the user type as manager.

**5.4 Accuracy**

The accuracy of the system is a significant issue since the test harness should provide the correct results of the code that is being tested. If the test harness gives the result as "Pass" then the code should be correct and function as per the requirement while if the test harness claims failure for a certain test case, then there should actually be some defect or bug in the code.

**Solution:**

Accuracy in the test harness can be ensured by adopting the following coding measures:

- i) Correctly parsing the XML test request document and decoding all the library names accurately.
- ii) Proper maintenance of library lists. Clearing the list every time the test harness runs.
- iii) Retrieving the correct paths for the dynamic link libraries from the repository.
- iv) Loading the test request libraries properly into the child app domain and recording the test results correctly for each test case.

**5.5 Logging**

Logging is a critical issue for the test harness since we need a efficient mechanism to send messages to the database and back to the demo client and display messages to the user. Messages should be test results or log files to the database.

**Solution:**

An interesting way of logging messages that has been adopted in this project is by recording the results of the test cases in a dictionary and print the values to the user after every test request is executed After all the test requests are executed we can retrieve a string by the getLog() method and create a log file by adding the details. We can then insert the log file in the database.

**5.6 Security**

Security is an important aspect of this system since it comprises of many different types of users and every user has different levels of functionalities associated with the system. The developer uses the test harness for testing their own code. The managers use it for checking logs. The QAs use it to test the developer's code. Lastly the TAs and the instructors use it to evaluate the requirements of the system.

**Solution:**

The system can be made secure by giving permissions to the users in such a way that the developer cannot access the functionalities of the manager.

**5.7 Extending to Project 4**

We will be further extending this project to a more complex version of the test harness where we would be implementing the following features:

- i) Running the test harness on multiple threads
- ii) Executing test requests simultaneously on multiple machines through a client-server model using Windows communication server (WCF)
- iii) Establishing a two-way communication between the primary application domain and the child domain.
- iv) Handling complex test code to enable that it can be run on the test harness
- v) Designing a graphical user interface (GUI) for the better usage and efficient interaction with the system.

## 6. References

- 1) <http://www.ecs.syr.edu/faculty/fawcett/handouts/CSE681/lectures>
- 2) [https://en.wikipedia.org/wiki/Test\\_harness](https://en.wikipedia.org/wiki/Test_harness)
- 3) Project starter code given by Prof. Jim Fawcett
- 4) *Joseph Albahari, Ben Albahari, O'Reily, C# 6.0 in a Nutshell*

## 7. Appendix

### 7.1 Sample Test Request XML File


Below is a sample test request in the form of an XML file which we will be providing as a command line argument to the user in the test executive.

```
<?xml version="1.0" encoding="UTF-8"?>
- <testRequest>
  <author>Moneesha Modi</author>
  - <test name="First Test">
    <testDriver>td1.dll</testDriver>
    <library>Library-1.dll</library>
    <library>Library-2.dll</library>
  </test>
  - <test name="Second Test">
    <testDriver>td2.dll</testDriver>
    <library>LibraryInterface.dll</library>
  </test>
</testRequest>
```

### 7.2 File Manager Prototype

The file manager prototype takes an input test request in the form of an XML file consisting of names of various .dll files. We parse the file using the XDocument class and get the various dynamic link libraries in the XML file. These libraries are then searched in the repository which is a directory tree and their entire DLL name along with the full directory path in which they reside is displayed on the console.

Below is the output of the file manager prototype:

 C:\WINDOWS\system32\cmd.exe

```
library : ../../../../FileManagerPrototype\DLLs\Library-1.dll
library : ../../../../FileManagerPrototype\DLLs\Library-2.dll
library : ../../../../FileManagerPrototype\DLLs\LibraryInterface.dllPress any key to continue . . .
```

### 7.3 Child App Domain Prototype

In the child app domain prototype, we create a child app domain, load the dynamic link libraries, create an ITest interface which loads the test driver and the associated test code and executes the code.

Please find below the output of the child app domain prototype:

```
C:\WINDOWS\system32\cmd.exe
Application Domain Demo #2
=====

Starting in AppDomain AppDomainDemo.exe

mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089
TestHarness, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null

AppDomainDemo.TestHarness
loading: "..\..\..\Tests\CodeToTest1.dll"
loading: "..\..\..\Tests\CodeToTest2.dll"
loading: "..\..\..\Tests\TestDriver1.dll"
loading: "..\..\..\Tests\TestDriver2.dll"

testing TestDriver1
Production Code: first being tested
test passed
testing TestDriver2
Production Code: second being tested
test failed

Press any key to continue . . .
```