

TEST HARNESS

Operational Concept Document

CSE-681 Software Modeling and Analysis
Fall 2016 - Project #1

By Venkata Santhosh Piduri
SUID:944740835
Instructor – Dr. Jim Fawcett
Date: September 14,2016

1 Table of Contents

1. Executive Summary	3
2 Introduction	4
2.1 Obligations	4
2.2 Organizing Principles and Key Architecture Ideas	4
3 Uses	5
3.1 How will users use the application?	5
3.1.1 Course Instructors & Teaching Assistants	5
3.1.2 Developers	5
3.1.3 Managers (Project #4)	5
3.1.4 Managed services team	5
3.2 New Features to implement in Project#4	5
4 Structures	5
4.1 User Interface	5
4.1.1 Test Request tab	6
4.1.2 Query about results tab.	7
4.1.3 Console	8
4.2 High Level Architecture of Real Test Harness System	8
4.3 Module or Package diagram of Test Harness	10
4.4 Activity diagrams	11
4.4.1 System Activity Diagram	11
4.4.2 Fetching Results from MongoDB Activity Diagram	13
5 Critical Issues	14
6 Prototypes	15
6.1 File Manager Prototype	15
6.2 Application Domain	16
7 Appendix	17
7.1 File Manager Prototype	17
7.2 Application Domain Prototype	18
8 Reference	20

1. Executive Summary

Generally, development of Large Software Systems needs a large set of developers and Quality Assurance members. They use various collaboration tools during their development and production release life cycles. They involve in a lot of collaborative activities to manage the versions of code changes and finalizing a production ready code by having various testing activities on master code branch. Some of environments that help in controlling versions of code are CVS, RTC, GitHub, etc. Before code going to production environment, members in Quality Assurance and Developer team starts conducting various types of test on master code repository. Based on results correctness they take necessary actions to make it stable. During this process if there is a testing application, which perform various types of test automatically, it will reduce a lot of time spent on manual effort.

The purpose of this document is to provide implementation details of automatic testing application for developers, which is called as **Test Harness**.

Current version of Test Harness will be primarily used by Course Instructors and Teaching Assistants to test and evaluate its functionalities, and for Developers to develop, maintain and support the Test Harness software system.

This document provides information about various concepts used in application, explains high level architecture, how packages are interacting with each other, uses of the application, about activities and actors involved in application, solutions for critical issues and prototype code of some utility packages.

Test Harness application is developed by following modularity. Some of the major packages used in the system are as below.

- a. Test Executive
- b. Test Harness Controller
- c. File Manager
- d. Application Domain, Loader
- e. ITest
- f. XML Parser

Following are some critical Issues that should be considered during development of Test Harness.

- a. Security
- b. User Interface
- c. Complexity of an application
- d. Handling multiple requests
- e. Handling dynamic linked libraries

2 Introduction

Test Harness is an automated testing tool that runs tests on multiple packages. Following are some obligations, organizing principles and key architecture ideas.

2.1 Obligations

1. Test Harness should support continuous integration to build code repository.
2. Application should support automatic testing for running many tests efficiently.
3. Should be able to accept test requests in XML format.
4. While testing application should be able to provide isolated environment for each test request.
5. Should provide ITest interface for all users, helps user to write test cases of an input application.

2.2 Organizing Principles and Key Architecture Ideas

1. For this project we will use .Net Framework-4 features and Visual Studio 15 for developing Test Harness application.
2. Entire architecture is designed using Model View Controller design pattern where model is separated from view and controller, view is separated from model and controller, in the same way controller is separated from view and model, so that any changes to any module will have very minimal code changes in other modules.
3. For Project #4, structure will be designed in a way that application will be able to handle multiple requests at the same time. In order to implement this feature, Producer-Consumer thread pool design pattern will be adapted.
4. Make use of AppDomain feature in .Net Framework that allows us to create isolated environment for each test request.
5. Make use of LINQ XML for reading and parsing of test requests and use of .Net I/O features to search dynamic linked libraries and copy them to temporary directory.
6. Blocking Queue will be used to hold test requests from user.
7. MongoDB is used to store test request results. With this data will be available for querying about test requests results and analyzing project stability.
8. Connection Pool concept is used to get MongoDB connection instance, so that we can ensure that all the connection instances are used efficiently.
9. Logging mechanism implementation helps in debugging application easily.

3 Uses

3.1 How will users use the application?

For Project #2 Test Harness application is used by Course Instructors and Teaching Assistants, Developers, Managed services team, and Managers.

3.1.1 Course Instructors & Teaching Assistants

Course Instructors and Teaching Assistants will use the application to test and evaluate all functionalities and inform the developer if any changes are needed. They also verify whether application is feasible and scalable to extend its features.

3.1.2 Developers

1. Developers have to go through this operational concept document and must implement all the requirements specified in document.
2. Developers can make use of Test Harness to test functional behavior of their code.
3. They are also responsible to extend features of application.

3.1.3 Managers (Project #4)

Managers make use history of results from Test Harness Application to know more about stability of project. They can query for getting test case results.

3.1.4 Managed services team

1. Main role of these users is to continuously support the application.
2. Should be able to provide temporary fixes for application if necessary until next release of the application.
3. Should collaborate with developers for any major fixes in application.

3.2 New Features to implement in Project#4

In Project #4

1. The design of this project can be extended to handle multiple requests in parallel.
2. User interface can be extended to multiple users.
3. Create a production environment to deploy Test Harness Application to serve multiple requests from users through Internet.
4. Apart from the above features in Project#4 we are going to introduce logging mechanism. It will be more helpful at the time of maintaining application.

4 Structures

4.1 User Interface

User interacts with Test Harness using GUI Interface. Layout of GUI is designed in tabbed fashion. Following are two tabs.

1. Test Request tab – In this tab user can make test requests and view results.

2. Query about results tab – In this tab user can query about the results based on date.

4.1.1 Test Request tab

Test Request tab is primarily used by Teaching Assistants, Course Instructors, developers and QAs. Following are some features of this tab.

1. Interface contains test request browse option, select test drivers to perform testing option, and viewing results option.
2. Users must use browse feature to give location of test request file.
3. After giving test request file location, all the names of the test drivers present in test request will be displayed as checkbox followed by name.
4. Test Request contains information about author, repository location, test drivers, code to test and its dependencies.

The screenshot shows a window titled "Test Harness" with two tabs: "Test Request" (active) and "Query about results".

In the "Test Request" tab, there is a "Test Request" label next to a text input field and a "Browse" button.

Below this is a section titled "Select test drivers to perform tests" containing a list box with three items: "testdriver1.dll", "testdriver2.dll", and "testdriver3.dll". Each item has a checkbox to its left. The checkboxes for "testdriver2.dll" and "testdriver3.dll" are checked. There are scroll bars on the right side of the list box. Below the list box is a "Submit" button.

At the bottom of the window is a "Results" section with a dashed border, containing a text area with the following text:

```
Total number of tests
Number of test fails
Number of test passes
```

Figure 1

5. Sample Test Request file will be as follows.

```
<?xml version="1.0" encoding="utf-8" ?>
<testRequest>
  <author>Santhosh</author>
  <test name="First Test">
    <testDriver>testdriver1.dll</testDriver>
    <library>tc1.dll</library>
    <library>tc2.dll</library>
```

```
</test>
<test name="Second Test">
  <testDriver>testdriver2.dll</testDriver>
  <library>tc2.dll</library>
</test>
</testRequest>
```

6. User can select test drivers for a specific input application to perform testing as shown in Figure 1.
7. User can submit request by clicking submit button.
8. After submitting input, Test Harness will respond to user by giving information about number of tests executed, and number of tests succeeded, and failed tests.

4.1.2 Query about results tab.

Managers use this tab for checking stability of the project.

1. By choosing a date, manager can request test harness for all test request results on a particular date.
2. After picking date, user must click on submit button to send request as shown Figure2.
3. Test Harness will process requests and respond to user by displaying all test results on a requested date.

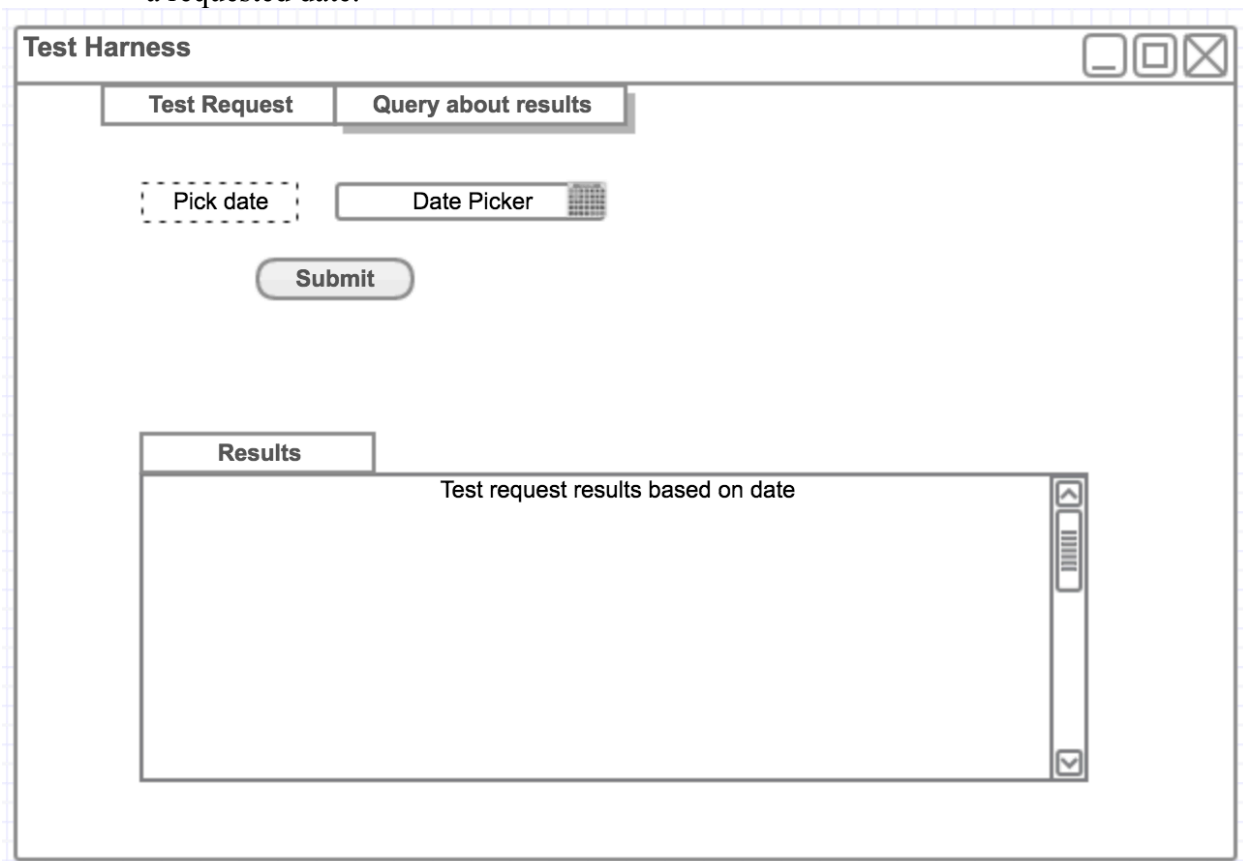


Figure 2

4.1.3 Console

All the logging information related to Test Harness will be displayed on console of an application as indicated in Figure 3

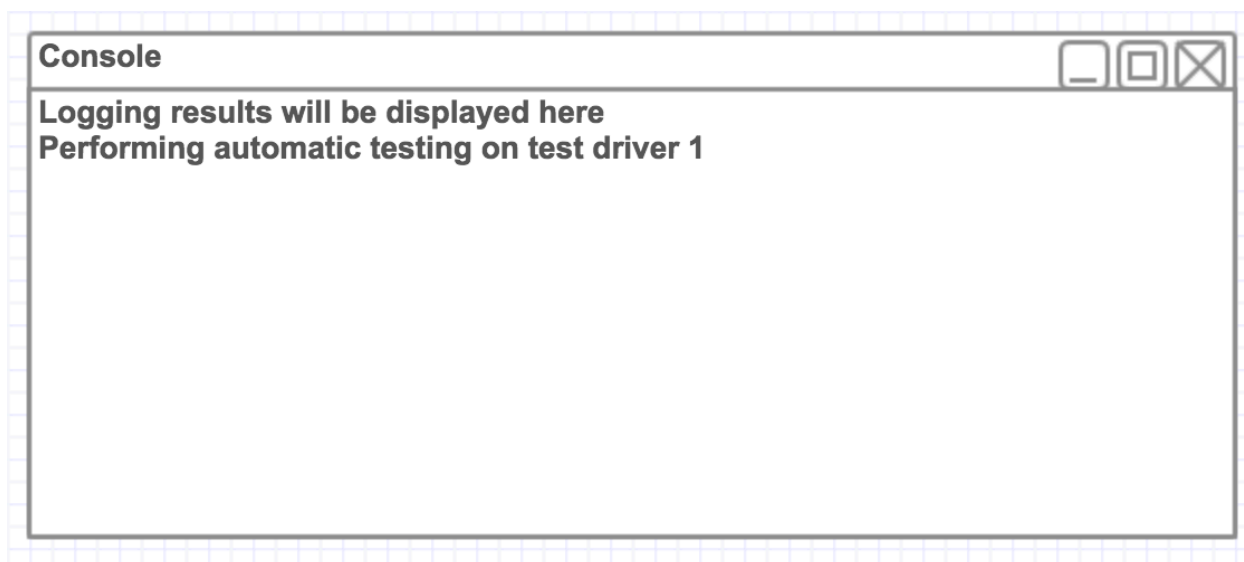


Figure 3

4.2 High Level Architecture of Real Test Harness System

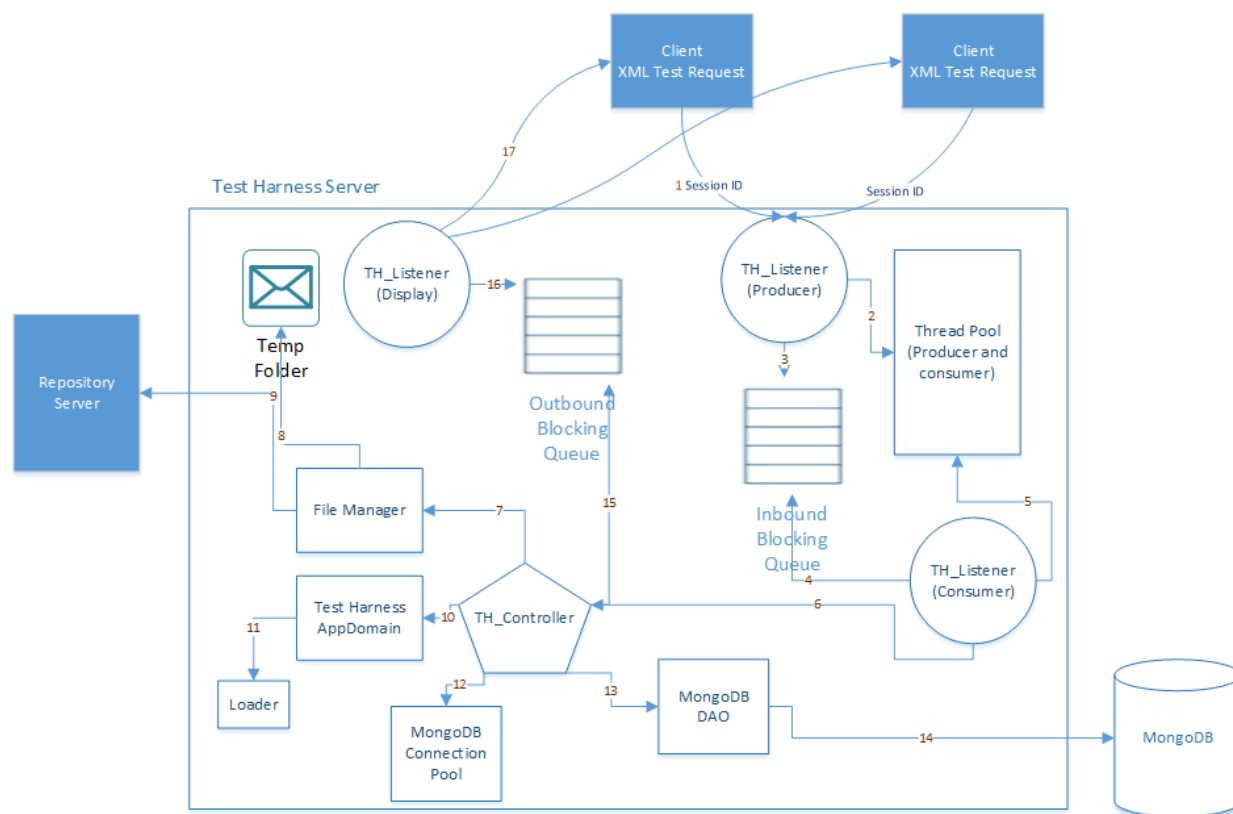


Figure 4

Figure 4 depicts the high level architecture of Test Harness application.

1. User requests Test Harness server to perform testing by uploading test request in xml format as input.
2. Multi Thread mechanism in Test Harness is implemented by adapting **Producer-Consumer Thread Pool design pattern** where producer acts as enqueueer for incoming requests to inbound blocking queue and consumer acts as dequeuer from inbound blocking queue, and it will process request. After processing the requests, consumer will keep final results to outbound blocking queue.
TH_Listener (Producer) – keeps on listening for client requests
After it receives request from client, listener will make use of producer thread from thread pool.
3. Producer thread will push request to inbound queue.
4. TH_Listener (Consumer) – keeps on listening to inbound blocking queue.
5. If Listener finds any pending requests in queue, then listener will de-queue the request and assign it to consumer thread in thread pool.
6. Internal implementation of Test Harness is designed by following **MVC design pattern**, where there will be
 - a. Model – which holds data
 - b. View – User Interface
 - c. Controller – Which controls whole applicationTH Controller will act as controller; it will call required component in respective packages to get work done.
7. Controller make use of utilities in File Manager package to perform file related operations
8. File Manager retrieves test drivers, application to be tested and its dependencies from Repository Server.
9. File Manager copies all retrieved files from repository server to temporary folder in Test Harness.
10. Controller call AppDomain which intern uses Loader to inject DLL files to process.
11. Loader make use of reflection concept to get meta data information about dynamic linked library files.
12. Controller will get MongoDB Connection Instances from connection pool.
13. Connection Instance is passed to MongoDAO package which intern uses instance to perform database related operations.
14. MongoDAO inserts all testing results to MongoDB
15. Controller also inserts all logs to outbound queue
16. TH_Listener (Display Listener) keeps on listening to outbound queue if it finds any item in queue, then immediately it de-queues the item.
17. And listener respond to particular client by sending items present in outbound queue.

But for Project #2 we will go with simple architecture (standalone console application), where it contains blocking queue which stores incoming requests, controller setup which process requests, and local folder act as repository.

4.3 Module or Package diagram of Test Harness

Test Harness is designed in modular fashion and also adapted MVC design pattern. By following MVC pattern dependencies between packages will get reduced. Following are some modules used in test harness system

All Test Harness related modules are wrapped in global namespace called Test Harness

1. Test Executive Module

Test Executive contains main entry flow of Test Harness. This module will be able to accept input from user and request will be enqueued in Blocking Queue.

2. Controller Module

1. This module will act as dequeuer from blocking queue, and responsible for creation of AppDomain for each test request and initiates testing flow.
2. Following are some advantages we will get by running DLL files in AppDomain.
 - a. AppDomain create isolation process for each DLL so that it will continue processing when there is code execution failure.
 - b. Gives security for processes created by AppDomain.

3. XML parser Module

XML parser make use of LINQ XML .Net Framework features to perform operations on XML file (Test Request from user). It parses whole test request input and convert to application readable format.

4. File Manager Module

Following are list of File Manager functionalities.

- a. Search for dynamic linked libraries in a specified path given by AppDomain Handler.
- b. Display names of all DLL files present in specified path.
- c. Copying all DLL files from specified path to temp folder in Test Harness.

5. Loader Module

Loader uses reflection feature provided by C# .Net to load all assemblies and inject them to AppDomain.

6. MongoDB Module

1. This module is to provide available MongoDB connection instance to Controller from connection pool. Following are some advantages of using connection pool.
 - a. Minimize creation of new connection object.
 - b. Effective usage of connection objects
 - c. Reduces of number stale connection objects

MongoDB DAO is one of the class in this module contains all database related operations, some of the operations are

- a. Create or insert – adds new document to collection.
- b. Update – modifies document in collection.
- c. Read – retrieve documents from collection.
- d. Delete – delete documents in collection.

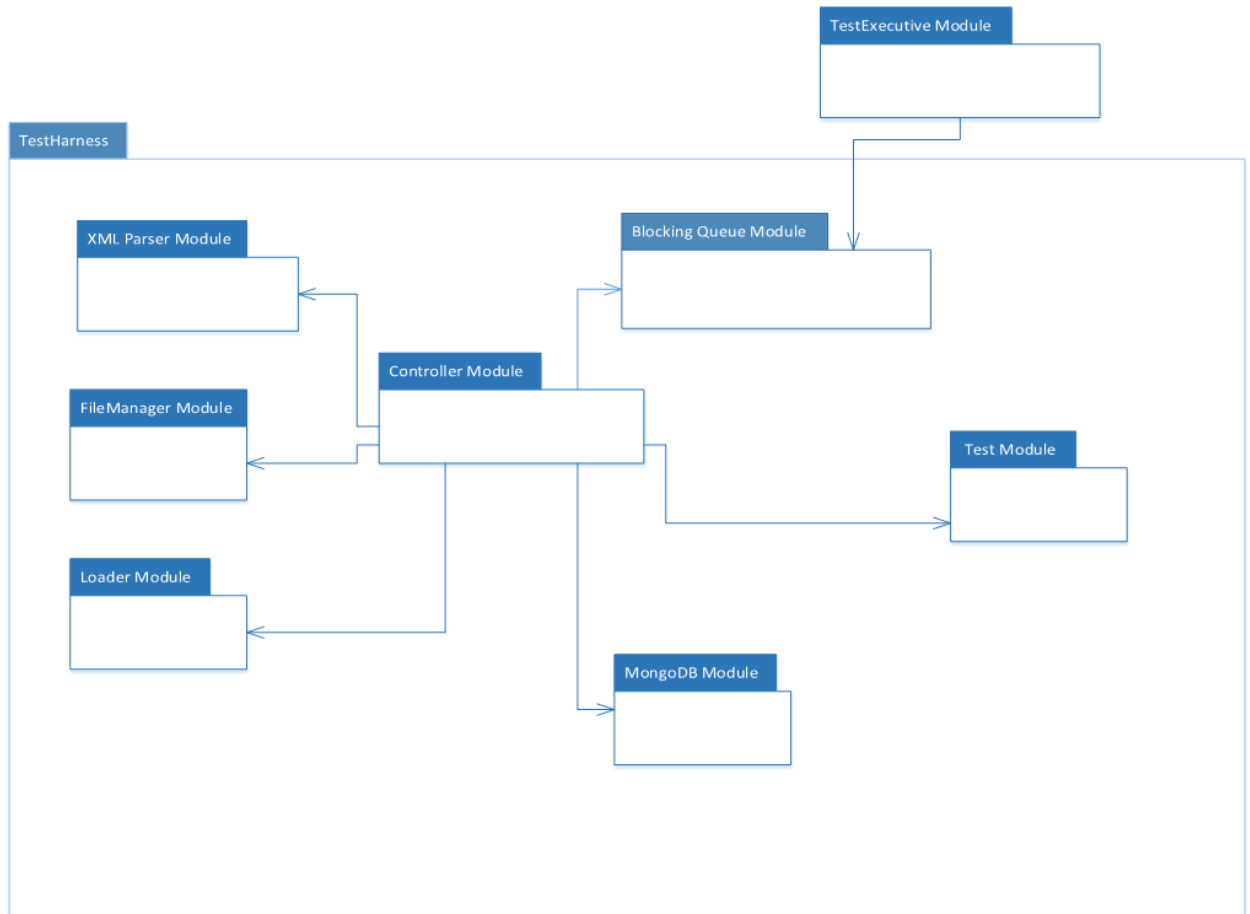


Figure 5

7. Test Module

This module contains ITest interface, used by users who are writing test cases for their project. ITest interface declares two methods - one is test() it takes no arguments and returns bool value, and next one is getLog() method which returns log information in formatted way. Developers should implement ITest interface and write test cases.

4.4 Activity diagrams

4.4.1 System Activity Diagram

Figure 6 depicts the system activity diagram.

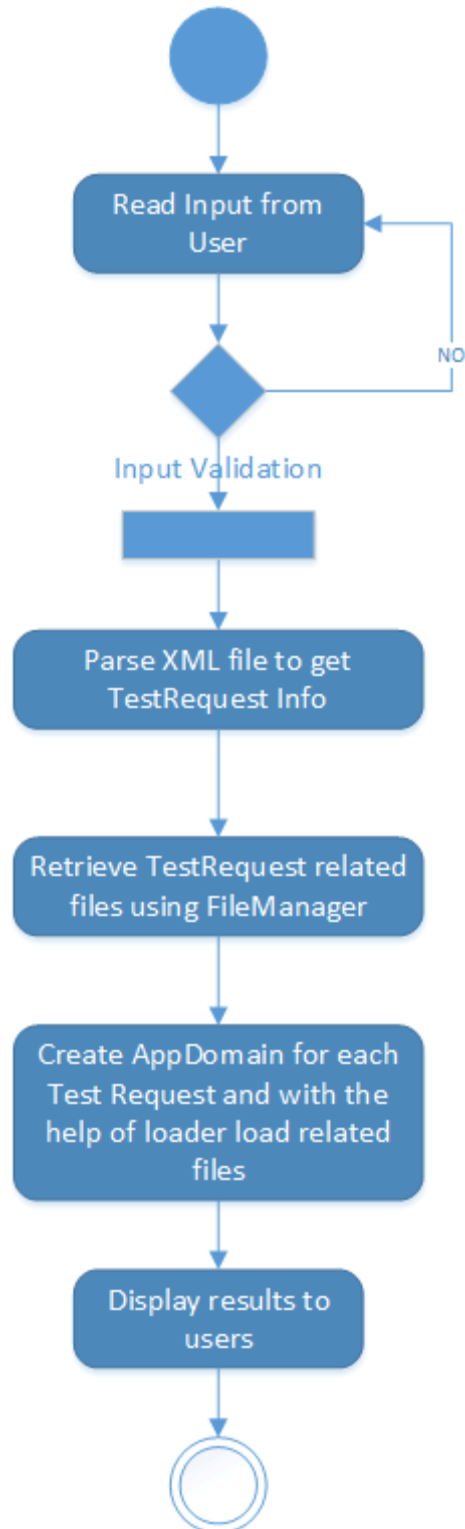


Figure 6

1. Read Input from user – this action is to read input requests from users, input file will be in XML format.
2. XML parser is used to parse test request and convert to application readable format.

3. File manager will retrieve all test request related files from repository server and copies to temp folder.
4. Controller will create child application domain for each test request and child application domain starts processing the request with the help of loader.
5. Child application domain returns testing results to controller, and controller display functionality will display all the results on console.

4.4.2 Fetching Results from MongoDB Activity Diagram

Following activity diagram depicts the fetching results from mongo database flow.

Who are the users of this use case?

Managers, Team Leaders, Teaching Assistants and Instructors are users of this use case.

High level Activity flow:

1. User requests for testing logs.
2. Test harness process the request by getting connection instance object from connection pool and passing it to MongoDB Data Access Object package (Database query related operations resides) to retrieve requested information.
3. Displays results to users.

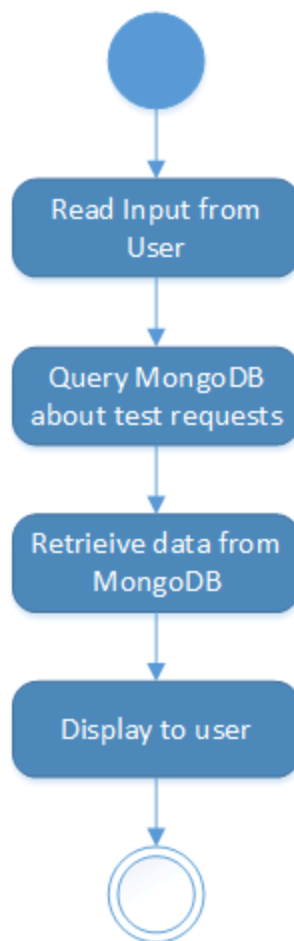


Figure 7

5 Critical Issues

1. Issue #1

In project #2 user will be giving test request in the XML format. How can user know what xml tags should be used so that his request goes smoothly?

Solution: Test harness will provide XSD file to all users. XSD defines structure of an XML document, which elements and its attributes are allowed in XML. So that users who are developing test request XML file should include XSD file.

Following are some advantages of XSD

- During compile time itself user can able to figure out what is wrong with his request file.
- Can able to define data type, and restriction on data.

Sample XSD for XML test request will be as follows

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="NewXMLSchema" xmlns:tns="NewXMLSchema"
elementFormDefault="qualified">

<xsd:element name="TestRequest" type="tns:TestRequest"/>

<xsd:simpleType name='testname'>
  <xsd:restriction base='xsd:string'>
    <xsd:pattern value='\w{10}'/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="library">
  <xsd:sequence>
    <xsd:element name="codetotestlibrary" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="tests">
  <xsd:sequence>
    <xsd:element name="test" minOccurs='0' maxOccurs='unbounded'>
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="testdriver"
type="xsd:string"/>
          <xsd:element name="library" type =
"tns:library"/>
        </xsd:sequence>
        <xsd:attribute name='name' type="tns:testname"/>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="TestRequest">
```

```

<xsd:sequence>
  <xsd:element name="repositoryLocation" type="xsd:string"/>
  <xsd:element name="author" type="xsd:string"/>
  <xsd:element name="test" type="tns:tests"/>
</xsd:sequence>
</xsd:complexType>
</xsd:schema>

```

2. Issue #2

Performance: Test Harness is using Mongo Database to store testing results. In order to communicate with database, application have to create mongo database connection instance. Since too many users use Test Harness, there will be too many connection instances. Obviously there will be performance issue when creating new connection instances when there are too many connections and communicating with database, how application is going to handle this scenario?

Solution: Application is handling the above scenario by using MongoDB Module, where pool of connection instances and pool configuration will be maintained. Following are some advantages with connection pool.

- a. Reduces connection instance creation time
- b. Reduces number of stale objects.
- c. Efficient use of resources.

3. Issue #3

Security: Test Harness will be used by too many user having different roles, for instance user may be Developer, Manager, and QA. How application is restricting users to perform only his own activities?

Solution: Each and every user who uses test harness will be given username and password authentication information. User have to include his credential information along with test request. Application will cross verify user's credential information and process the request.

4. Issue#4

Assume user X and user Y are working in Project A and Project B respectively.

User X have written test driver (TD1.dll) to test his one of the code related to project A and User Y have written test driver (TD1.dll similarly name unknowingly) to test his project B code. As test harness is multithreaded application, accidentally two users X, Y are using test harness at same time, how test harness will handle same named test drivers from different projects?

Solution: Test Harness will get meta data information from repository server and create a directory structure based on projects and copies test request related files in Test Harness server. Hash table is a key value data structure where key is automatically generated by taking project name appended with file name and value is fully qualified path of DLL file. This helps harness in searching dependency files in O (1) time complexity.

6 Prototypes

6.1 File Manager Prototype

Test Harness uses File Manager package in following ways.

1. To search for particular format of files in specified directory and subdirectories.
2. To display all file names.
3. Copy files from source to destination location.

1. Search Files

To implement Search functionality, can use .Net Framework System.IO package where all IO operation related classes resides. For getting all names of files in a specified director can use this method.

System.IO.Directory.GetFiles(string path,string searchpattern,SearchOption searchoption)

This method will accept specified directory, search patterns, and Search Option, which say where to search for files.

2. To display file names, can use *System.IO.Path.GetFileName* method that accepts fully qualified file name and returns file name with extension alone.
3. To copy files from source to destination, can use *System.IO.File.Copy* method which accepts source and destination directories.

Output for File Manager Prototype will be as in Figure 8

1. Input for this prototype will be specified directory location
2. Output will be list of all file names in a given location.

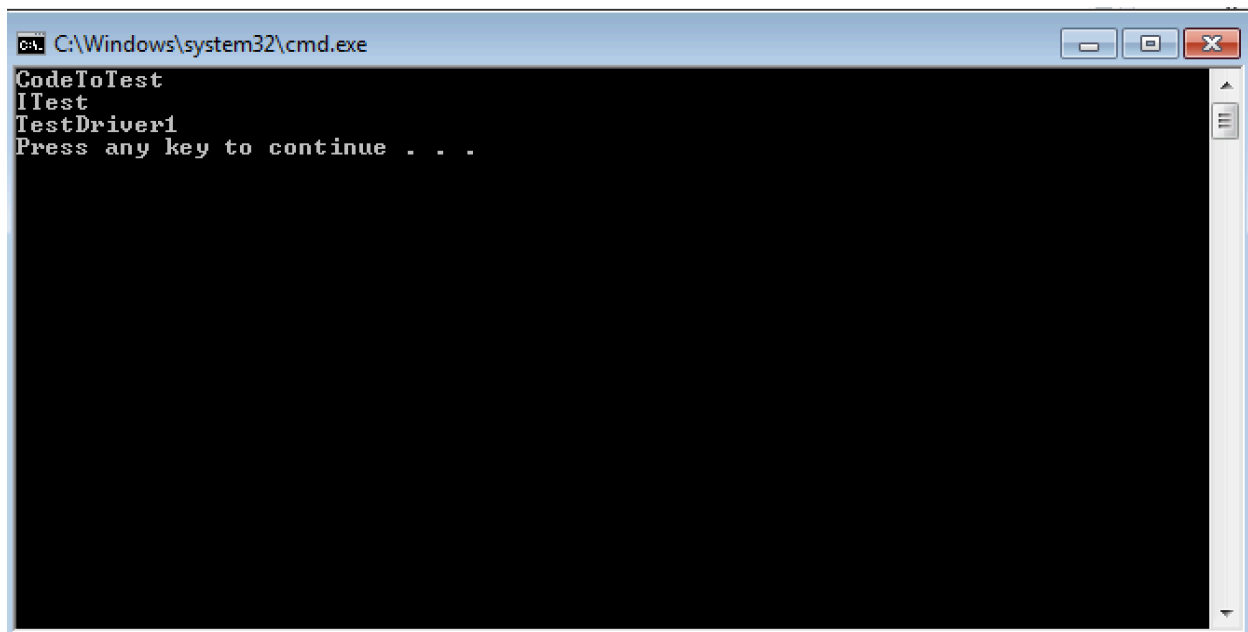


Figure 8

6.2 Application Domain

Following are some information about Application Domain I have learnt while developing prototype.

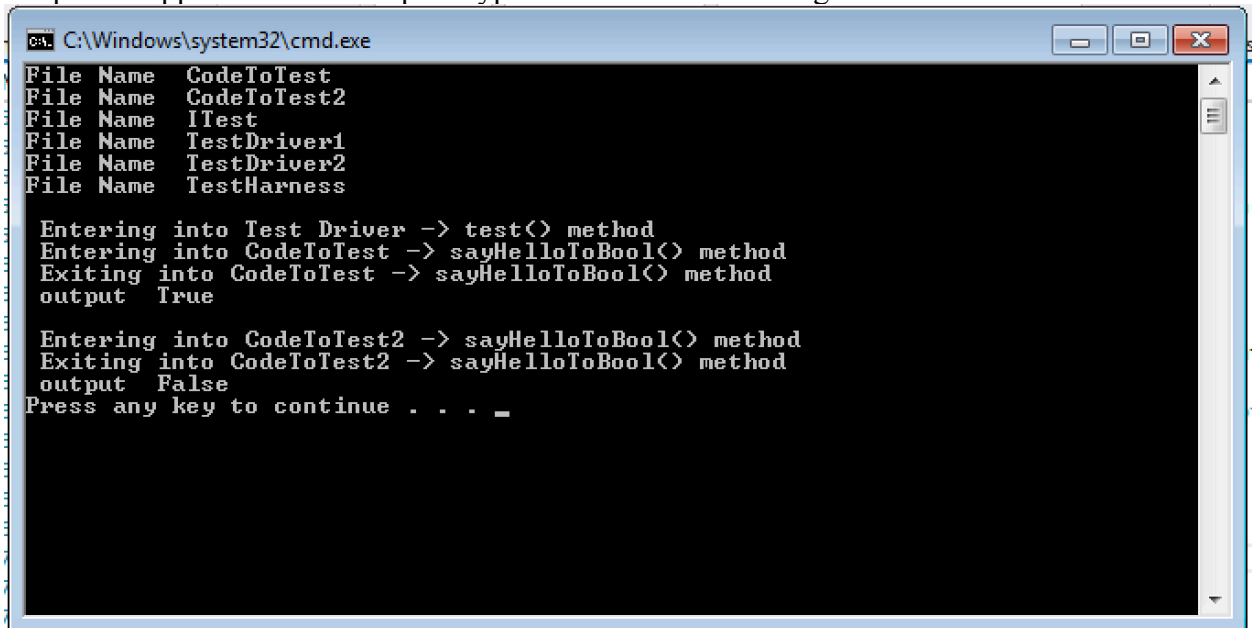
- Application Domain is used to create separate isolated container within the process.
- Test Harness uses Application Domain to run test request in an isolated environment to preserve security, and to unload libraries. Application Domain uses reflection to load libraries and to call methods present in loaded libraries.
- Application Domain will accept libraries, which are residing only in application base path.

Following are some important concepts helps in creating application domain

1. AppDomainBaseSetup is used to setup configuration information for newly created application domain.
2. Evidence class is used to create evidence for newly created appdomain
3. Assembly is one of the reflection class used to perform reflection operation such as loading libraries by giving filename.

In prototype it was coded to demonstrate creation of child application domain and to demonstrate how assembly is used to load of libraries (DLL). Child application domain will execute test method present in loaded library and display the result.

Output for application domain prototype will be as shown in Figure 9



```
C:\Windows\system32\cmd.exe
File Name CodeToTest
File Name CodeToTest2
File Name ITest
File Name TestDriver1
File Name TestDriver2
File Name TestHarness

Entering into Test Driver -> test() method
Entering into CodeToTest -> sayHelloToBool() method
Exiting into CodeToTest -> sayHelloToBool() method
output True

Entering into CodeToTest2 -> sayHelloToBool() method
Exiting into CodeToTest2 -> sayHelloToBool() method
output False
Press any key to continue . . . _
```

Figure 9

7 Appendix

7.1 File Manager Prototype

Main aim of File Manager package in Test Harness is to give all file related operations such as searching for a particular file in specified location, displaying files present in given location, copying files from source to destination location.

File Manager interface will be as follows.

```

public interface FileManager
{
    /// <summary>
    /// This method is used to retrieve all file names present in given directory.
    /// will accepts directory location and which file formate you are looking for
    /// </summary>
    /// <param name="directory"></param>
    /// <param name="wildCardFileExtention"></param>
    /// <returns>array of fully qualified file names in given directory</returns>
    String[] getFilesInSpecifiedPath(String directory, String
wildCardFileExtention);

    /// <summary>
    /// This method display all the filenames without extention
    /// </summary>
    /// <param name="files">array of files</param>
    void displayFileNames(String[] files);

    /// <summary>
    /// this method will copy files from one directory to another diectory
    /// </summary>
    /// <param name="source">Source directory information</param>
    /// <param name="destination">Destination directory information</param>
    /// <returns></returns>
    bool copyfiles(string source, string destination);
}

```

FileManager interface provides with three method declaration for getting files, displaying files and copying files. One of the test harness class called THFileManager class implements FileManager interface methods.

Test Executive for File manager prototype will be as follows

```

//Getfile name in fully qualified name
FileManager fileManager = new THFileManager();
string[] files =fileManager.getFilesInSpecifiedPath(@"../..dll", "*.dll");
fileManager.displayFileNames(files);
//copy files from source to destination
fileManager.copyfiles(@"../..dll", @"C:\help\temp");

```

7.2 Application Domain Prototype

Application domain setup is used to create configuration information for new child domain. Configuration information contains application base path, private bin path, private bin probe path, and application name information. Below is snippet of code show of application domain setup class.

```

AppDomainSetup domaininfo = new AppDomainSetup();
domaininfo.ApplicationBase = AppDomain.CurrentDomain.BaseDirectory;
// domaininfo.PrivateBinPath =

```

```
Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "dl\\");
// domaininfo.PrivateBinPathProbe =
Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "dl\\");
```

Evidence is used to set information that constitutes input to security policy decisions.

```
//Create evidence for the new AppDomain from evidence of current
Evidence adevidence = AppDomain.CurrentDomain.Evidence;
```

Child application domain is created by using application domain CreateDomain static method.

```
// Create Child AppDomain
AppDomain domain = AppDomain.CreateDomain("ChildAppDomain",
adevidence, domaininfo);
```

```
// creates instance
```

CreateInstanceAndUnwrap method is used to create instance and assembly is used to load dll files using loadfrom method. And with the help of reflection test method is executed.

```
AssemblyProxy proxyInstance =
(AssemblyProxy)domain.CreateInstanceAndUnwrap(
    typeof(AssemblyProxy).Assembly.FullName,
    typeof(AssemblyProxy).FullName);
    string[] files =
fileManager.GetFilesInSpecifiedPath(AppDomain.CurrentDomain.BaseDirectory,
    "*.dll");
    //fileManager.displayFileNames(files);

    foreach (string file in files)
    {
        //Loads DLL files and returns Assembly object reference
        Assembly assembly =
proxyInstance.getLoadedAssembly(System.IO.Path.GetFileName(
    file));

        // AssemblyName[] arrayOfAssems =
assembly.GetReferencedAssemblies();
        //with the help of Reflections program can able to go through all
types
        //if any class implements ITest interface it will create instance
and call test method

        Type[] types = assembly.GetExportedTypes();
        foreach (Type t in types)
        {
            ITest.ITest
            lib = null;
```

```
        if (t.IsClass && typeof(ITest.ITest).IsAssignableFrom(t))
            lib = (ITest.ITest)Activator.CreateInstance(t);
        else
            continue;
        Console.WriteLine(lib.test());
    }
}

//used to unload domain
AppDomain.Unload(domain);
```

8 Reference

1. http://www.w3schools.com/xml/schema_intro.asp
2. https://docs.oracle.com/cd/B28359_01/java.111/e10788/intro.htm
3. <http://www.ecs.syr.edu/faculty/fawcett/handouts/cse681/>
4. <https://docs.mongodb.com/manual/crud/>