Remote Test Harness

SUID: 56789-3036

Name: Himanshu Gupta

Syracuse University

CSE: 681 Software Modeling and Analysis Instructor: Dr. jim Fawcett

Operational Concept Document

Project 3

DATE: 26TH OCT 2010

Contents

[1. Executive Summary 4](#_Toc275892223)

[2. Introduction 4](#_Toc275892224)

[3. Requirement Summary 5](#_Toc275892225)

[4. Architecture of the System 6](#_Toc275892226)

[4.1 Context Diagram Description 6](#_Toc275892227)

[4.1.1 Test Harness Client: 6](#_Toc275892228)

[4.1.2 Test Harness Server 7](#_Toc275892229)

[4.2 High Level Architecture Diagram 8](#_Toc275892230)

[4.2.1 Description 8](#_Toc275892231)

[4.2.1.1 Server Description 8](#_Toc275892232)

[4.2.1.2 Client Description 9](#_Toc275892233)

[5. Application Activities 9](#_Toc275892234)

[5.1 Activity Diagram 10](#_Toc275892235)

[5.2 Description of Activities 11](#_Toc275892236)

[5.2.1 Client Activities: 11](#_Toc275892237)

[5.2.2 Server Activities 11](#_Toc275892238)

[6. Partitions 12](#_Toc275892239)

[6.1 Client Partitions 12](#_Toc275892240)

[6.1.1 Description of Client Modules: 13](#_Toc275892241)

[6.2 Server Side Partitions 17](#_Toc275892242)

[6.2.1 Description of Server Side Modules: 17](#_Toc275892243)

[6.3 Common Type used by Developer and Test Server 19](#_Toc275892244)

[6.4 Communication Methodology 21](#_Toc275892245)

[6.4.1 Communication between the Test Harness Server and the Client application 21](#_Toc275892246)

[6.4.2 Communication between Application Domains 23](#_Toc275892247)

[6.5 System Interactions and Request Handling 23](#_Toc275892248)

[7. Actors 26](#_Toc275892249)

[7.1 Developers 26](#_Toc275892250)

[7.2 Testers 26](#_Toc275892251)

[7.3 Architects 26](#_Toc275892252)

[7.4 Production Support/Maintenance: 26](#_Toc275892253)

[8. Uses 26](#_Toc275892254)

[8.1 Software Architect 27](#_Toc275892255)

[8.2 Software Developer 27](#_Toc275892256)

[8.3 Quality Assurance 27](#_Toc275892257)

[8.4 Deployment Team 27](#_Toc275892258)

[8.5 Maintenance Team 27](#_Toc275892259)

[8.5 Common Uses 27](#_Toc275892260)

[9. Critical Issues 28](#_Toc275892261)

[9.1 Displaying large results 28](#_Toc275892262)

[9.2 Deadlocks and Race Conditions 28](#_Toc275892263)

[9.3 Network Failure Issues 29](#_Toc275892264)

[9.4 Memory Logging of test cases 29](#_Toc275892265)

[9.5 Duplicate Files in same Domain/Test-suite 29](#_Toc275892266)

[9.6 Crashing of Child-App Domain 29](#_Toc275892267)

[9.7 Utilization of Resources 30](#_Toc275892268)

[9.8 SLA Relevance 30](#_Toc275892269)

[9.9 Performance 30](#_Toc275892270)

[10 Conclusion 30](#_Toc275892271)

[11 References 31](#_Toc275892272)

# **Executive Summary**

Remote Test Harness is a testing tool which is used to perform automated testing. The whole system’s architecture is based on Client-Server Architecture. Because of this it is possible to access the test server from remote locations.

Key Features of the Remote Test Harness application are:

* Multithreaded with each test suit in different App domain.
* Capability of serving multiple client applications and request.
* Efficient way of writing test functions by keeping the production code separate.
* Logging Facility.
* SLA Performance Monitoring.

The Client Application is capable of providing following functionalities

* Easy way to provide configurations for testing.
* Output in clean and human readable form.
* Functionality to *Stop* the testing process if required and to restart it.
* Ability to make dll libraries of the source files.
* Functionality to check status of the server.

The main set of uses of this application will be **developers** and **testers** who have the maximum involvement in the SDLC process. Developer can use this application to test, that the code changed or developed by him doesn’t breaks any other functionality in the persisting code. Testers may use this application to test the degree of stability of the application before staring manual testing process.

Some critical issues were identified for this application like possibility of confronting deadlocks and race conditions, network issues etc. It is very important to analyze the impact of these issues and their possible solutions.

# Introduction

Large Software Systems are composed of thousands of modules which interact among themselves or external system to provide the expected functionalities. This results in high impact of change on one module because of other. Some design patterns are followed to keep this cohesion and coupling minimum but it is impossible to make a module work in isolation. Because of this there is always a high risk of breaking other code because of change in one module.

A software application is only qualified as useful when it is capable of providing the specified functionalities in a reliable and stable manner. Software testing is done almost continuously throughout the development life cycle and makes it possible to test the quality of the software so that it can be used. Organizations may adopt any process of software testing among the available ones like feature testing, load testing, functional testing etc. Many of these require human interaction when testing and thus it becomes very time consuming and expensive process to test large software application. A failure of a single test case can waste a large number of man hour’s effort and engenders the requirement of testing it again.

To alleviate the problems described above it is essential to streamline the process of testing in such a way so that the complete testing of the software system is done at minimal cost. This Remote Test Harness tool is designed to serve this purpose by automating the testing process. Its purpose is to support frequent testing throughout the development process for both medium and large projects. It runs the tests contained in the modules and generates the log reports of the same which can be used by the user.

# **Requirement Summary**

Remote Test Harness must provide the following functionaries:

* It should be able to take a set of input files/assemblies so that it can run test on them.
* If multiple projects are given for testing at the same time then the system must use multithreading to test them simultaneously.
* It should provide data generation and data logging facilities, so that the test developers don’t have to repeatedly create that entire infrastructure for each of their test suits.
* It should also provide some mode of communication for clients to interact.

# Architecture of the System

The application as a whole communicates with the outer world for specific reasons like taking input, displaying results, communicating to external systems etc. The context diagram provided below demonstrates its interaction with outer world precisely. The whole system is divided into two main segments: client and server, so they need some network support to communicate with each other.

**Context Diagram**

## 4.1 Context Diagram Description

The above diagram represents the interaction of the system with the outer world like OS resources, other sub systems, other independent applications etc. Each application part knows only about the exposed public interface of other part and nothing about the internal structure of the application. The detailed explanation of the context diagram is given as under:

### 4.1.1 Test Harness Client:

This system is responsible to deal with the user input and the test harness server. It hides all the low level details of collecting file references, dll, other configuration settings from the user, compiling dll from the source files etc. The following system to which the Test harness client interacts are discussed below:

#### Build System

This system does the task of collecting the files and other library files whose path are provided by the user. This system then compiles them to make dll of them. These libraries are stored at some specific location at the client so that they can be accessed by the other system of this application.

#### User Input/Output

This system holds the task of taking the input and displaying output to the user.

#### File System

The whole system revolves around the source code files which are saved on the disk. The client interacts with the file system to read/write the files.

#### Network Resources

To communicate with the server it needs to establish a connection, check the files received that they are not corrupt etc. For all these tasks the harness client system makes use of network resources to communicate with the test server.

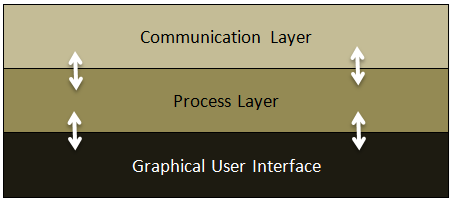
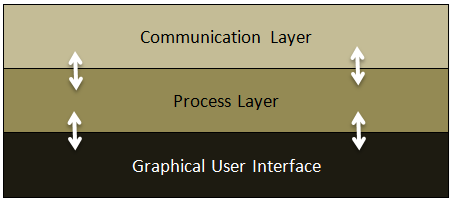
### 4.1.2 Test Harness Server

This system does the main task of executing the test methods present in the libraries. While executing these test cases results are logged in the format specified by the user. The logs are written in some file which is stored on the disk. To perform the write operations it interacts with the ***file system*** of the OS. This system is capable of accepting multiple request from the same or different clients and executive them concurrently.

## 4.2 High Level Architecture Diagram



Remote Server



Client 1 Client 2

## 4.2.1 Description

The High Level Architecture Diagram depicts that what each system does and to which layer the other layer can communicate. Above diagram also shows that the server application can talk to more than one client simultaneously.

### 4.2.1.1 Server Description

Test Harness Server can be categorized in three layers according to the functions provided by it at a higher level.

**Tester Layer:** This layer is focused on handling all the processes which are related to perform test, logging results, managing child application domains etc. So this layer acts as a core of this system.

**File System Services Layer:** This layer handles the tasks related to make new folders to store the files provided by the client, store the log file and deleting file which are of no use after testing.

**Communication Layer:** Thislayer interacts with the clients and handles all the communication part. The benefit of using this layer is this that the other layers of the system are not aware about it and hence any changes in the layers will have no or minimum impact over the functionality provided by the other layers.

### 4.2.1.2 Client Description

Client Application has a layered architecture where each layer has some defined responsibilities. This helps in keeping the concerns separate and the code becomes easy to manage. Each layer can be divided into sub layers or partitions depending upon the need and complexity. A brief discussion is given below about each layer.

**Graphical User Interface Layer:** This layer provides a set of controls to the user so that he can interact with this system. User can provide the configurations needed by the test harness system to perform the testing. After a user provides the setting/configurations then it communicates to the *Process Layer* to perform further processing at the client side.

**Process Layer:** This is the middle layer which communicates with GUI Layer and Communication Layer. It is responsible for the generation of dynamic link libraries using the provided source files. It also prepares an XML file based on the settings provided by the user so that it can be used by the communication layer to pass it to the test server. It also maintains the data objects used by the GUI for the display purpose.

**Communication Layer:** This is the layer which handles the communication part. It gives the files; information needed by the test server and also receives the information from the server. It also handles the security part, detection of network issues, state of the test server etc.

*Note*: These layers communicate to the layers which are just above or below them and not to any other layer. So GUI cannot directly talk to the Communication Layer and vice versa.

# Application Activities

A task given to the software system can be grouped as a set of activities. Each of these activities may be divided into further smaller activities. Complete knowledge of activities is required to build a focused software system which serves to the requirements in a precise manner.

## Activity Diagram

This diagram demonstrates the activities with higher level partition on the level of likeliness. Furthermore these activities can be divided into two subgroups named client side activities and server side activities. The activities start when user gives input to the system and ends when the system displays the output after processing the given input. Each of the activities is discussed as under.

 **Activity Diagram**

## 5.2 Description of Activities

### 5.2.1 Client Activities:

All of these client activities happen at the client side and is initiated by the user. A much detailed explanation is given below:

**Define or Revise Module Code**: This is the basic step which is done by the developer. Any application is made up of one or more modules. The functionalities provided by these modules are tested using the test harness framework. Once a module is developed it may need some changes in future to fulfill the requirements. Each time the module code is changed it engenders the need to test the whole system to ensure that there are no blockers in the application.

**Define Derived Logger if Needed:** Developer who is writing the code can define the derived logger to provide the logging functionality if required. Logging is considered as the essential part of the application which proves to be very helpful in debugging or identifying the issues.

**Define Derived TVG if Needed:** Here TVG stands for Test Vector Generator which is used to provide automatically generated test data used in automated testing. The developer may write code to read the test data from some file or to generate it on the fly based on some logic. It can also contain the predicted output to validate the test results.

**Define Test:** This test function is the one which will be executed to validate the test. Developer may use any of the provided members like TVG, Logger etc in the test function. All of the functionalities to be tested may be written in some private function which will be called from this test function.

**Provide config details + path to Modules + Libs:** The developer has to provide some details like name of the project/test suite he is going to give for test, the location of the modules, also the libraries used by the module, the order of testing etc.

**Build Libraries and add to Test harness:** This activity compiles the modules and then packs them in dll format. After that they are transferred over the network to the test harness server with the other configuration provided by the user. The information provided by the user is compiled in XML format which acts as a common language to both the server and the client.

### 5.2.2 Server Activities

After receiving all the required information and files the server starts its process of testing. The process is broken in small activities which are described below:

**Start Test Harness Process:** The server receives the files on which tests are to be run with additional configurations and validates it using the configuration file sent by the client.

**Update Display:** The user/developer has to be notified about the errors or successful upload of the input files. This step is done by giving a response to the client in XML format.

**Setup Environment:** This activity is composed of smaller activities which are dependent on each other. The application starts a new thread which starts a new app-domain where the testing will take place. Each project will be tested in a new child app-domain by a separate thread. This is done to keep the main test harness thread up and running even if the other child thread are killed or crashes because of some exception or something else. The Loader class Object is loaded so that it can be used for performing the additional tasks required for testing.

**Load Dll and add to Loader Collection:** All the provided dll files are loaded one by one in the child app-domain and are also added to the Loader’s collection. This collection maintains reference to all the loaded dlls. A List can be used here as a collection obkect.

**Sort Loader Collection:** The user may provide the order in which the testing has to be performed. To achieve this, the collection in Loader class is sorted on the basis of defined order.

**Run and Log test:** The loader then runs the test method if it finds the *ITest* type in the types fetched from the loaded dll. The logging is also done as which test function of which module is being tested, how much time it took and was it a success or failure? This whole information is logged in XML format and when all the tests are run then it is written to a file which is saved on the disk.

If the implementation is complete then the process is not repeated otherwise if some more changes are incorporated then the whole process is repeated until the implementation is not complete.

# Partitions

An application as a whole is able to do a lot of functionalities but all these functions can be grouped together to make partitions which are easy to maintain and simple to grasp. This software system is implemented using the layered architecture in which each layer has a specific functionality. These layers consist of various other partitions which make it possible for the developer to focus on a specific task and to keep the concerns apart. On the same pattern the identified modules of Remote Test harness are described below.

## Client Partitions

The client communicates with the server application to perform the instructions given by the user provided by using graphical user interface. This application when seen from a developer perspective can be segmented in a number of modules where each module has some responsibilities and to achieve this, it communicates with other modules. The identified modules are gives as under:

1. Application Controller
2. GUI
3. Data Controller
4. File Controller
5. Build Controller
6. Proxy Controller
7. Update Display Controller
8. XML Util



**Modular Diagram for Client**

### 6.1.1 Description of Client Modules:

**Application Controller:** This module is the main module which starts all other processes. It makes the GUI ready so that the input can be provided by the user.

**Graphical User Interface:** This is the module which presents a graphical interface to the user so that he can provide the input. After accepting the input from the user it interacts with the Data Controller. If some validation error is found then the user is informed about it. The data model properties are binded to some of the fields of the GUI.

GUI is divided in two main panels.

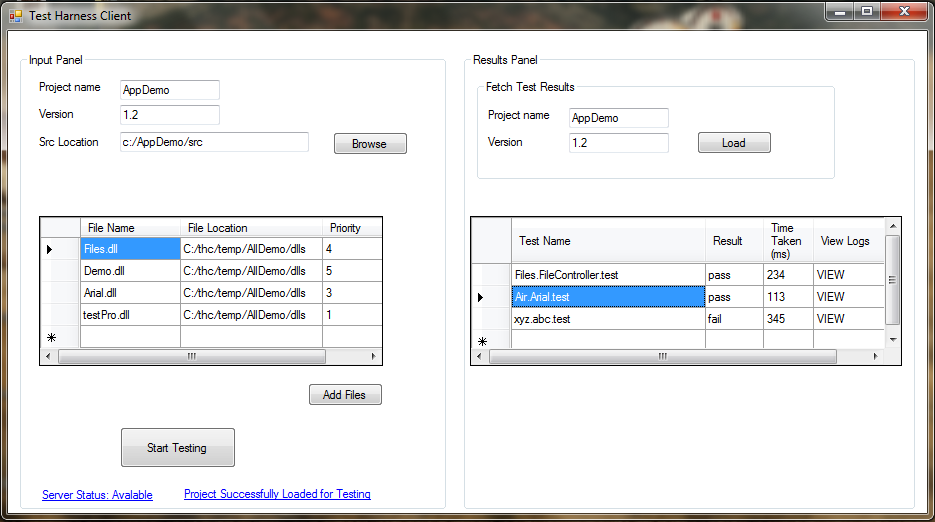
**Input Panel:** This panel is used to display the fields needed to get the details required for the project to be tested. These details are then converted into an XML file which is given to the server. Files are loaded separately after sending the xml file and getting the ready message from the server.

When the user provides the src path the client fetches the entire source file and compiles them to make dll of them. These dlls are then saved on the disk so that they can be transferred. The dlls formed are displayed in the dataset shown below. The user can also add additional files to it using the Add Files option. The priority field is editable and the user can provide the values by editing it.

The user can also set the priority in which the testing will occur of the following modules. The higher the number is higher will be its priority. It means the module having priority 5 will be executed before all other modules having priority less than 5. This priority information is also included in the XML config file which is used by the server.

This part also shows the current status of the server. It is a Hyperlink which enables user to click on it. When the user clicks it, it checks that the server is listening or not. If it gets a reply then it is displayed as working otherwise ***Inactive*** is shown instead of displaying ***active***.

After user clicks on the **Start Testing** button**,** the communication with the server starts. If everything goes fine than the user is displayed “*Project Successfully Uploaded for Testing*” otherwise error message is displayed.



**User Interface as shown to the user**

**Output Panel:** Theoutput panel is used to display the test results to the user. The use has to first provide the test project name and its version. Then the server finds the log file, if a file is found then the file is returned and the data grid is populated by parsing it. If the server fails to find a file representing the given parameters then an error message is displayed to the user.

There may be a case in which user tries to fetch the log file of the test project which is being executed by the test server. In this case the user is notified that the project is currently being tested.

The View Logs Column of the data table is a hyperlink. If a user clicks on it then he is shown a child window which contains the detailed log of that test.

**Data Controller:** It deals with the validation of the data coming in and going out to the GUI. It uses a model named Data Model and a utility named *XML Util* to perform its task. Data Model stores the data needed by the File Controller, details given by the user etc. XML Util is also used by the Data Controller to make the XML file, using the information given by the user. This XML file is transferred over the network to the server.

The data model object will collect all the user provided file paths in a List. This is used by the File Controller to get the reference to the files provided by the user.

**File Controller**: File Controller is used for the following purpose

* To collect the files from the locations as specified by the user. This list is stored in the Data Model and the File Controller uses this Data Model Object to fetch the files. If this controller any issue in validating it then the user is notified about it.
* All the correct verified paths of the files are stored in the **FileModel** Object. This object too has a **List** which is used for this purpose.
* Checks for files with the same name, if there at different locations provided by the user. If found it notifies the user.
* Checks for the valid paths of the given files.
* Updates the file model after refining the information provided by the user.

**Build Controller:** This controller does the task of compiling and packing the source files into dynamic link libraries (dll). The source and other file information to be transformed into dll are taken from the File Model Object. It compiles and stores the dll files on the disk at some location. The information about the dlls as how many dlls are there and where are they stored are stored in the DLL Model Object. This model also contains the information as what all cs files belong to which dll file.

**Proxy Controller:** This controller does the task of handling the Proxies. This controller is used by the Application Controller to talk to the other remote services provided by the other server applications. Using this controller makes it possible to use multiple proxies, if needed with a layer of abstraction to hide the details. Also it handles the errors that can be thrown by a proxy.

**Proxy:** This is the main class that interacts with the service hosted at some remote server. It is used by the Proxy Controller.

**XML Util:** This Module is used to make an xml representation of the input details provided by the user. This module is also used to parse the XML response received and the logs in XML format. This module is used by the Data Controller.

**Update Display Controller:** This controller is used to achieve the functionality of updating the UI as per the information received from the server. This controller after getting response from proxy controller communicates with Data Controller which makes use of XML Util and updates the Data Model and thus View is updated.

## Server Side Partitions



**Modular Diagram of Test Harness Server**

### 6.2.1 Description of Server Side Modules:

**Application Executor:** This will be the main module which will handle the initialization of the application. This loads the configuration needed to bring the application up. This can also be used to provide the configurations needed for security purpose.

**Service Contract:** This module defines the services which will be available for the clients to use. This module only defines the interface that will be open to use and is independent of the implementation of these defined services.

**Service Impl:** This module defines the implementation of the services defined by the Service Contract. This module may further define some services used by it but those will not be the part of the Service Contract. To add a service to use remotely it has to be defined first in Service Contract and then implement it in ServiceImpl.

**File Controller:** This is used by the Service Impl to save the received file from the client, make appropriate folder to store them, delete them after use. It is also used by the Logger to archive the old log files. All this information received from the client is processed and is stored in a *Config Model* object which is used by the Loader. The config model has full information about the test suite which may be needed by the *Tester* for testing purpose. This config model also has a reference to File Model which holds the reference to the files to be tested.

**XML Util:** This utility is used by the Service Impl to parse and understand the information passed by the client in an XML file. This is also used to send messages to the user in XML format which are in Object representation at the server side. (Like transforming a detailed log of a module from txt to XML format).

**Loader:** This module loads the dlls to be tested and executes the test functions if it finds a particular type. This module also logs the time taken by each test method to execute. This information can be used by the user to determine that the service being tested adheres to the SLA or not? It also logs the results while executing these tests and saves them in XML format. The location of loader is known to the Tester module. The tester module loads the Loader in the child domain so that it can do the loading and testing stuff.

**Tester:** This module handles the following responsibilities:

* Starting a new Thread
* Creating a new App-Domain
* Loading Loader
* Unloading app-domain

**Starting a new thread:**

Tester starts a new thread for each request of testing a new test-suite. This thread knows about the configurations provided by the user and does its task in an independent fashion.

**Creating a new app-domain:**

The newly created thread creates a new child app-domain providing the settings needed like application name, reference to evidence, private path etc. All the testing of the test-suite is done inside this app-domain. Using a new app-domain provides two main advantageous:

1. It keeps the information about each test suite apart.
2. If one test-suite execution crashes then it has no impact over the other running test-suites in other app-domains. This is a nice way to keep the application up and running even if the test-suite crashes. The client can request the information about the crashed test-suite. The logs can be used by the user to identify the cause of the crash.

**Loading Loader:**

Loader is the most critical part of the test harness system. It is itself loaded in the app-domain by the tester and then it loads all the provided dlls in the app domain one by one. Then it checks for the libraries which are provided by the user in the app-domain and checks for a particular interface/type. If it finds a type of that then it invokes a method whose information is known by the Loader. The test results are logged using Logger Module.

**Unloading App-Domain:**

After the Loader finishes executing all the test cases contained in test-suite it informs its status to the main app-domain tester class which unloads the child app domain.

## 6.3 Common Type used by Developer and Test Server

The test server is designed to work in such a way that it searches for a defined type which is known to it. As long as it finds that type it is capable of executing the test function contained in it. So to use this test-suite should make use of this common type.

**ITest**: This is the common type whose information is known to the server and client is obligated to implement this interface if he wants the server to be able to test the module developed by him. The server application searches for this particular type and knows that it has a method named ***test()***which takes no argument and returns a Boolean value.

A complete explanation is provided below about this common type which is used by developer:

**Test Suit:** Test suit is the collection of the assemblies with a single configuration file. This configuration file contains the information like test suite name, number of files, version, ip of the client, date and time when request is sent, time zone of the client machine, logging option, logger to use, dependency of modules in which testing has to be done etc. This information is used by the test harness server to initialize and do the testing process. The test server only knows about ITest and it tries to find it in the collected types. Once found the test() method is invoked using reflection and the results are logged.

A test function is discoverable only when the structure is followed as specified. The main types used while developing a test are discussed as under:

1. ITest
2. ILogger
3. ITestVectorGenerator

 **Class Diagram of Test Structure used by Harness Server to test**

1. **ITest:**  This interface defines a test function which returns a Boolean value and takes no argument. The test class which is designed to test the module will implement this Interface and the body of the test function will call the other functions which will test the functionality of the module part by part and will return a Boolean value which represents that the test result is a pass or fail.

A developer has to define the order of the internal test functions which are called from the body of the test function if needed. The implementing test class also has a reference to the ILogger which is used by the developer to log the results, exceptions, process etc. of the test functions. The data needed as in input by the tests is retrieved from the ITestVectorGenerator whose reference is also maintained by the test class.

A property named title is used to know the title of the running test function.

1. **ILogger:** This is the interface which is defines two functions named write() and showAll(). Developer has the option to use any of the loggers named: File Logger, Console Logger, and Memory Logger. Each of these Loggers has a specific logging ability.
2. **File Logger:** It logs the information to a file.
3. **Console Logger:** This Logger logs all the information to the console.
4. **Memory Logger:** This Logger can be used to store the test results of one function so that it can be used by the other function. For this purpose it maintains a Dictionary which maintains the qualified name of the test function and its output.

The user can choose what kind of logging he needs and can specify it as an input to the client. This information is used by the Test harness Server to log the results.

1. **ITestVectorGenerator:** Theinterface defines a method named *GenerateNext()* whose purpose is to generate the input arguments required by the methods developed to test the functionalities. The data which is generated may be loaded from some txt file or it may be generated using some logic at the run time.

The developed code using all the types mentioned above in correct manner makes a Test Driver which can be tested by the Test Harness Server. He should also make sure that all the resources needed by the test() method of the test driver are initialized correctly in the constructor on some initialization block or before using them. Failing to do so can result in some run time exception. Exception handling should also be used wherever there is a possibility of some unexpected behavior.

## 6.4 Communication Methodology

Communication in this proposed system can be categorized in two main segments.

1. Communication between the Test Harness Server and the Client application.
2. Communication between Application Domains.

Each of the above communication is discussed below in detail.

### 6.4.1 Communication between the Test Harness Server and the Client application

The remote server is hosted on some server which is used by the clients located at the different geographical locations. These client applications communicate to the test server using some protocol.

This application architecture uses Windows Communication Framework of .NET 4.0 for its communication. The server application exposes its services using web services. Using web services allows developers to build assemblies containing types that can be accessed using simple HTTP. Using this also increases the interoperability of the system as all the communication happens over HTTP using XML (SOAP) messages. So any client on any platform like Java on UNIX, Python on Solaris can easily communicate with the server. It increases the ROI of the system and lessens the pain of replicating the same system in different languages for different platform.

Though WCF provides many binding options but the best one which serves the requirements is **WSDualHttpBinding.** Using this makes the application capable of having two-way conversation.

**Security:** It is very important to make these services secure as if they are left open then there exist some risk of security breach or some attacks. To ensure that the system services are provided only to the valid clients there should be some security. To achieve this WS-security will be used which covers many aspects of security, including digital signatures, authentication and encryption of SOAP messages. A client request is served only when it clears the security check.



Python Client on UNIX

Java Clienton UNIX

.NET Clienton Windows

WS Proxy

WS Proxy

WS Proxy

Test Server XML Web Services

**XML Web Services provide high level of portability**

Using web services introduces some performance issue as all the communication happens in XML format and at every end there arises a need to parse the XML messages to and from some Object. This can take some significant time on a very busy server which handles large amount of data.

### 6.4.2 Communication between Application Domains

The application needs some mechanism to communicate between the child and parent app domain. This is done in order to provide the configuration to initiate the testing and also to share the logs information.

To achieve the system uses the remoting framework of .NET . The Objects being passed from one app domain to other using reference should be serializable. The remoting framework provides an excellent support to communicate between app domains by marshaling and unmarshaling the objects by ref or by value.

ds

Cross Domain

Tester

Loader

Primary Add Domain

Child Add Domain

**Cross Domain Communication**

## 6.5 System Interactions and Request Handling

The system discussed so far clears that the interaction is happening between the remote test servers and the client applications. The aim of having the test application on server and making it multithreaded is to make it able to handle many clients at the same time.

To initiate the testing process the client makes a request. The communication for initiating a test process happens in two parts in the given order:

1. Client sends a XML
2. Client sends the resources to test

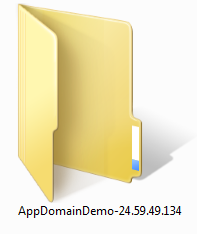


**Activity Diagram of client interaction with server**

1. **Client sends a XML**: This is the first step of letting the server know about the test-suite the client wants to be tested. This XML file has the meta data about the test-suite. Server uses this information to make the resources ready so that it can handle further communication. A sample format of this file will be somewhat like the one which is shown below.

|  |
| --- |
| <?xml version="1.0" encoding="utf-8" ?>  <test\_suite>  <name>AppDomainDemo</name>  <version>1.0.0</version>  <file\_logger>true</file\_logger>  <console\_logger>false</console\_logger>  <time>2010-11-05T08:15:30-05:00</time>  <client\_ip>24.59.49.134</client\_ip>  <!-- Assembly files information-->  <assemblies\_info>  <no\_of\_files>4</no\_of\_files>  <assembly\_file\_names>  <file\_info>  <file\_name>FileModel.dll</file\_name>  <priority>4</priority>  </file\_info>  <file\_info>  <file\_name>DemoMode.dll</file\_name>  <priority>2</priority>  </file\_info>  <file\_info>  <file\_name>DomainInfo.dll</file\_name>  <priority>1</priority>  </file\_info>  <file\_info>  <file\_name>xyz.dll</file\_name>  <priority>3</priority>  </file\_info>  </assembly\_file\_names>  </assemblies\_info>  </test\_suite> |

The server uses the information provided and makes a folder on its file system using the IP and the name of the test-suite to be tested. The format of the folder name is <test-suite-name>+<ip>.



The assemblies\_info tag is used for giving the information about the files which are transferred. It has the number of files, file names and also the priority in which they should be tested. The higher number shows the higher priority. In case two files have the same priority number then any of them can be tested in random fashion. However the testing will still follow the priority of other numbers.

This XML file is stored in this folder. The server also makes some more folders for keeping assembly files and log files. Using ip in the folder name removes the possibility of error in case two different clients send the same test-suite name. Using the number of files to be transferred and their names the server app makes sure that it has all the files needed to do the testing. In case there are some files missing the testing is not started and the client is notified about the problem.

1. **Client sends resources to server:** This is the step in which the assembly files are transferred. The server accepts them and saves them in the folder made for those assemblies. Here this step awaits for an OK signal form the server to send the resources. In case the server is heavily loaded then the server sends signal that it the test-suite cannot be tested at present time.

*Note*: All the folders and files saved at the server side are temporary and once the user is notified and fetches the results the logs/test-results may be archived or deleted as per user requirements. The server has to do nothing with assembly files once the testing is finished so it deletes them after the execution of the associated test-suite.

# Actors

Software systems are developed with a high focus on its requirements which serves its users. Failing to serve the same will result in its failure. The main actors of this software system will be the development team, testers and the integration team.

## 7.1 Developers

Developers may use this tool to test that their code is working or not. If all tests are passed then it means that the developed code is providing the assumed functionality. If some of them fail then the developer may see the logs/Exception of the failed test cases and can fix them to test them again. The process can be repeated a number of times as it doesn’t requires human effort for testing the

## 7.2 Testers

Testers may use this software as a preliminary test to verify that none of the functionality breaks. If all the test cases are successful then the manual testing can be started otherwise the log reports are shared with the development team so that they can fix the problem.

## 7.3 Architects

They can keep a check over the quality of the software system by regularly analyzing the test results of the application. It will also prove very helpful when deciding the SLA of some services.

## 7.4 Production Support/Maintenance:

Peopleworking in the production support team or in maintenance team will use this application to test that all the modules are working fine or not after every change is made.

# Uses

This application has a high usability in a project’s lifecycle. It may be used by software developer, architect, and testers to ensure the quality of the present code. Some of the uses which are identified as per role are describes as under.

## 8.1 Software Architect

**Quality Test:** Software Architect may use this tool to test that the application is working or not. If all the tests pass then there is a high probability that the apparition works fine. On the basis of these result the Software Architect can deduce that the application/module being tested is **stable and reliable** or not?

**Performance Monitoring:** The Software Architects can use this tool to test that the service defined in the modules adheres to the SLA agreement or not? This application is capable of logging the time each test function took. The GUI can be made able to get the details of the SLA in an XML format and show a comparison to the user of the actual time taken and the proposed one.

## 8.2 Software Developer

**Developer level testing of code:** Developers have the responsibility to compose test cases which may test the modules they have developed. The developer may use this tool to run those test cases which validates that the developed functionality is working or not? Moreover it also checks that the newly added code doesn’t break any other functionality of the application.

## 8.3 Quality Assurance

Quality Assurance team can use this tool to verify that the application they are going to check is fully functional or not. Running all the test drivers before starting of manual regression testing proves to be very useful as it saves a large amount of time and money. If any of the test drivers fails then there exists no point in conduction the manual regression testing. On the other hand if all the tests run successfully then there is a highly probability of facing to blockers in the application.

## 8.4 Deployment Team

Deployment Team may use this system to test that the complete system is stable or not. Even before deploying a patch the system has to be tested and verified.

## 8.5 Maintenance Team

Maintenance team may use this tool several times as they have the task of modifying the application to serve the business needs.

## 8.5 Common Uses

Moreover Remote Test Harness can be used in a number of ways for the testing which can prove very useful for many processes like QA advancement, Quality Audit etc. Some of its common uses are discussed below:

* + **Increased productivity due to automation of the testing process:** Automation of testing process can result in increased productivity as the mundane test cases can be automated. The time saved by this can be used to design smart test cases which can mimic the real world situation.
  + **Increased probability that regression testing will occur:** If the application’s all test case are passed then it gives a green flag for the regression testing. Failure of any of the test cases will show that the system is unstable and it makes no sense to test the system which is sure to fail.
  + **Increased quality of software components and application**: This tool helps in enhancing the quality of the software components and applications as using this testing starts at much earlier phase and that too in a repetitive fashion without any human involvement after starting the test server process.
  + Ensure that subsequent test runs are exact duplicates of previous ones.
  + **Proper utilization of resources and time**: This tool can do the testing when the office is not staffed (ie. at night). We can also make use of the same machine which is used for the development in the day time.
  + A test script may include conditions and/or uses that are otherwise difficult to simulate (load, for example)

# Critical Issues

## 9.1 Displaying large results

After running all the test drivers it becomes important to show the results to the users in some understandable format. The number of tests can vary from some hundreds to thousands depending upon the robustness of the project.

**Solution**

The GUI of the client should show the results in some logical order. The results may be grouped together on the basis of functionality, module, passed/failed or in alphabetical order. The user may click on the test name which will show the detailed log to the user.

## 9.2 Deadlocks and Race Conditions

As this system makes use of multithreading which solves problems with throughput and responsiveness, but in doing so it introduces new problems: deadlocks and race conditions.

**Solution**

Bypassing these problems requires careful programming. To reduce the complexity thread pools and queues can be used. This indeed engenders the necessity of proper design and thorough testing of the system in a simulated environment.

## 9.3 Network Failure Issues

As the core of the system (test harness) is located on some remote server so there are chances of confronting network issues. If the network failure happens in the mid of the assembly files transfer or during the callback to give the status back to the system it can halts the process or can give unexpected results.

**Solution**:

As it is very complex to deal with these kinds of issues as they are not predictable. A better approach of storing the results, state, name, IP and time can help to investigate and restart the process. Also the client application should have a status display of the server status that it is up and running or not. Server should host a service which returns some message for this purpose.

## 9.4 Memory Logging of test cases

This system provides a facility to log in the memory the test results of the test functions. Now this becomes really complex to decide for the developer as what results to put in the memory logger and what not to put.

**Solution**

Test class should have some hierarchy. Priority based testing can be used to solve this issue.

## Duplicate Files in same Domain/Test-suite

There is a possibility of having files of same names in the test-suite which are at different locations. This can cause a serious problem at the server side.

**Solution**

The test harness application should check that the names provided of the assemblies are distinct. If there are duplicates then an error message should be shown to the user.

## Crashing of Child-App Domain

Each test suite is tested in a separate application domain and a separate thread. If the test function throws an exception at the runtime which is not handled or caught then the app-domain crashes.

**Solution**

**.**NETprovides a handler to handle unexpected exceptions at run time. Also the developer should make sure that all the possible exceptions or the unmanaged code is under try catch block. This makes the application safe.

## Utilization of Resources

As the test harness application is at remote and there exists a possibility of getting too many requests for testing test-suits. A computer has limited resources and overloading that computer will hamper the performance of other processes.

**Solution**:

The application can be made to use a thread pool. On getting a test request the application checks that if it has a free thread or not in the thread pool. If it finds a free thread then it is used otherwise the application should check that the total number of current threads in execution is less than or equal to the maximum number of allowed threads. If it is less than a thread is made on the fly and is used by the application.

## SLA Relevance

This tool is capable to log the time taken by each test method and so it can be used to test that the functions adheres to the SLA or not? But the results from the test server may or may not suggest the real life data.

**Solution:**

The test server should have the same configurations as of the deployment server. This will increase the possibility of getting the figures much closer to the real conditions.

## 9.9 Performance

Performance has been always a major issue in software life cycle. This tool may have performance issues as there can be a number of client requests at a same time which can overload the server.

**Solution:**

A queue can be implemented which will be used to hold the request if the data traffic becomes too high. The System can then execute the request by pulling out those requests from the queue.

# Conclusion

* This system is maintainable as the complexity is low.
* System is flexible as we have kept the coupling between the modules minimum. This gives us the option to extend the system when needed at the minimum cost.
* *Feasibility of Architecture*: The architecture is feasible as all of the technical requirements can be served using any modern programming language framework.
* *Cost*: If the cost estimation is done of the project then the estimate will not go much high because the amount of coding required is not much. Moreover the maintenance cost will also be less because the complexity of this project is low.

# References

<http://searchsoftwarequality.techtarget.com/news/article/0,289142,sid92_gci1277008_mem1,00.html>  
<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.86.7291&rep=rep1&type=pdf>

<http://msdn.microsoft.com/en-us/library/1c9txz50.aspx>

<http://ondotnet.com/pub/a/dotnet/2002/11/25/copying.html?page=3>

<http://www.codeproject.com/KB/threads/ThreadingDotNet.aspx?msg=2557762>