TEST HARNESS

Operational Concept Document

CSE-681 Software Modeling and Analysis
Fall 2016 - Project #3
Instructor – Dr. Jim Fawcett

Venkata Santhosh Piduri
Email id: vepiduri@syr.edu
SUID:944740835
Date: October 19,2016

Table of Contents

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, for Developers to develop, maintain and support the Test Harness software system, and for Program Managers to know about test results done by developers and by other team members.

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 architecture is designed with two servers and an arbitrary number of clients.
The complete Test Harness application is functionally separated into three layers:

1. Test Harness layer
2. Repository layer
3. Client layer.

Each layer is implemented in modular fashion. Some of the major packages in each layer are as follows:

Test Harness layer:

a. Test Harness service contract – Test Harness expose its core services to clients by using service contract
b. Blocking Queue – It is a thread-safe queue and it is used to queue all incoming requests from clients and repository server.
c. Message Module – This module uses XML Link feature to create and parse messages.
d. File Manager Module – This module is used to handle all file related functionalities.
e. Storage & Logger Modules – These modules are used to store test results and logs in repository server and for logging test results.
f. Application Domain & Loader Modules – These modules are used to create new child application domains and load assemblies into App Domains.

g. Thread Pool – This module provides threads for test harness to process each client requests.

Repository layer:

a. Repository Service Contract – Repository server exposes its functionality to clients and Test Harness by using WCF Service contract
b. File Manager Module – This module is used to check requested file existence in a particular repository and helps in pulling files that needs to be uploaded by client.
c. DAO Module – This module used to communicate with mongo DB, to get information about file dependencies and test results information.

Client layer:

a. Client service contract – Clients exposes its services to both the servers by using this module.
b. GUI – This module is used to provide graphical user interfaces for Test Harness clients
c. HiResTimmer – This module is used to calculate latency between calls to server
d. Peer Connection Module – This module helps in creating channel and proxy objects.

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.
6. Test Harness should be able to expose its services using WCF, so that clients can make use of application by creating services proxy object.
7. Should be able to store all testing results in Repository server.
8. Should be able to retrieve assemblies/files from Repository server.
9. Repository server also should expose its services.
10. Test Harness should be able to run concurrent test requests from clients.

11. Client user interface should be designed and implemented using Windows Presentation Foundation (WPF).
12. Users Should be able to query Repository server for test results.

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 system is resided in Test Harness server, Repository server, and client machine. All automatic testing related functionality will be there in Test Harness server, Assemblies, Storage related functionality will be resided in Repository server, User Interface for Test Harness will be there in client machine.
3. Followed peer to peer communication architecture to communicate between each system mentioned above using Windows Communication foundation.
4. 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.
5. Test Harness system structure is designed in a way that application can be able to handle multiple requests at the same time. In order to implement this feature, one Producer-multiple Consumer thread pool design pattern is adapted.
6. Make use of AppDomain feature in .Net Framework that allows us to create isolated environment for each test request.
7. 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.
8. Blocking Queue will be used to hold test requests from user.
9. MongoDB resided in Repository server is used to store test request results. With this data will be available for querying about test requests results and analyzing project stability.
10. Connection Pool concept is used to get MongoDB connection instance, so that we can ensure that all the connection instances are used efficiently.
11. Logging mechanism implementation helps in debugging application easily.
12. User Interface is developed by following Command Pattern, where user click on tabs will be considered as command.
13. User Interface is designed by using Windows Presentation Foundation provided by .Net Framework.

3 Uses

3.1 How will users use the application?

For Project#4 Test Harness application is used by Course Instructors and Teaching Assistants, Developers, Quality Assurance team, Managed services team, and Project Managers, Sales team.

3.1.1 Course Instructors and Teaching Assistants

Course Instructors and Teaching Assistants will use the application to test and evaluate all functionalities in demonstration mode. And can review code and inform the developer if any

changes are needed. They also verify whether application is feasible and scalable to extend its features.

Impact on design:

As described above, Course Instructors and Teaching Assistants will use the application to test and evaluate all of its functionalities. Considering this situation, Test Harness application is providing demonstration mode to users. So that users can easily get to know what is happening behind the screen and can evaluate all functionalities in this mode.

3.1.2 Developers

Developers are primary users of Test Harness application. Two types of developers are involved in this category; Developers who implements Test Harness application and Developers who uses Test Harness to perform integration testing on their projects.

Developers who implements Test Harness application have the following roles.

a. Have to go through the Operational Concept Document and must implement all the requirements specified in document by considering performance and critical issues.
b. They are also responsible to suggest and extend new features of Test Harness application

Developers who uses the Test harness application have the following roles.

a. Can be able to upload test related files into repository server
b. Can be able to perform continuous integration testing on projects
c. Responsible to maintain stable base repository by making use of Test Harness

Impact on design:

As Developers are one of the main users of system. Test Harness application should give user friendly interface resulting increase in usage of Test Harness application. If user interface is bad, users find difficulty in interacting with Test Harness application. Test Harness application user interface is designed in Tabbed fashion which helps users to navigate easily through all the features provided by Test Harness.

3.1.3 Quality Assurance team

Quality Assurance team is one of the primary users who efficiently use Test Harness application. Team keep on testing project baseline to make sure that projects are meeting requirements. These users have almost similar role of developers, but the difference between them is the type of testing they perform.

Impact on design:

Testing huge application takes too much time, so at the end of the day QA team will queue up all the tests to be performed on projects, next day morning they get to know the overall results. Test Harness mainly concentrated on how QA team will queue up all the tests to be performed. If application is providing user interface to enter name of the test driver to perform testing, that will be time taking for QAs to enter each and every test driver names. In order to make this easy for QA team, application is providing file upload option where QAs needs to provide all file names in file and upload to test harness. And also providing testing whole project option in user interface.

3.1.4 Managed services team

Following are few important points how application will be used by Managed Services team

a. Main role of these users is to continuously support the application.
b. Should be able to provide temporary fixes for application if necessary until next release of the application.

c. Should collaborate with developers for any major fixes in application.

3.1.5 Project Managers

Project Managers uses Test Harness to get whole information about test results on particular project which helps to know about stability of the project.

Impact on design:

Application is storing all the test results in MongoDB database and providing a feature that helps users in querying test results. So that users can get to know whole information about test results.

3.1.6 Sales team

Sales team plays very important role in marketing Test Harness in market. Team will contact different organizations and present all the features provided by Test Harness and ask them to use the Test Harness application for better automatic continuous integration testing.

Impact on design:

Since main role of Sales team is to present all the features of Test Harness, application is providing rich user interface and also test executive to demonstrate all the features provided by Test Harness application.

3.2 New Features to implement in Project#5

In Project #5

a. Test Harness design will be extended to support C++ projects
b. Increasing repository server functionalities:
    - Maintaining code baseline for the projects
    - Dependency management
c. Implementing build server which builds and caches execution images and libraries as needed for test and deployment.
d. Check-in and check-out functionalities for users to check in, checkout source code into Repository server.

4 Structures

4.1 User Interfaces

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

a. Repository tab
b. Test Request tab
c. Query tab

4.1.1 Repository tab

Repository tab consists of all repository related features and used mainly by developers. One of the main features of this tab is uploading/ asking repository server to pull files, from clients. This is providing option called “Browse” to browse file directory to upload. Once file directory is browsed, all the files present in that directory are displayed under “select files to upload” category. User can be able to select all files or can be able to select particular set of files to upload to repository server. Figure 1 depicts Repository tab user interface.

While uploading files user has to provide dependency configuration file which is either in XML or in JSON format contains information about file dependencies.
Following is the sample JSON format dependencies configuration file.

```
{
	"basepath": "..\\project1\\testdrivers\\",
	"fileName": "TestDriver1.dll",
	"version": "1.0",
	"author": "santhosh",
	"description": "this is sample test driver",
	"dependencies": [{
		"basepath": "..\\project1\\codeToTest\\",
		"fileName": "codeToTest1.dll",
		"version": "1.1.1"
	}, {
		"basepath": "..\\project1\\codeToTest\\",
		"fileName": "codeToTest2.dll",
		"version": "1.0"
	}]
}
```

Uploaded results will be displayed under Results section.

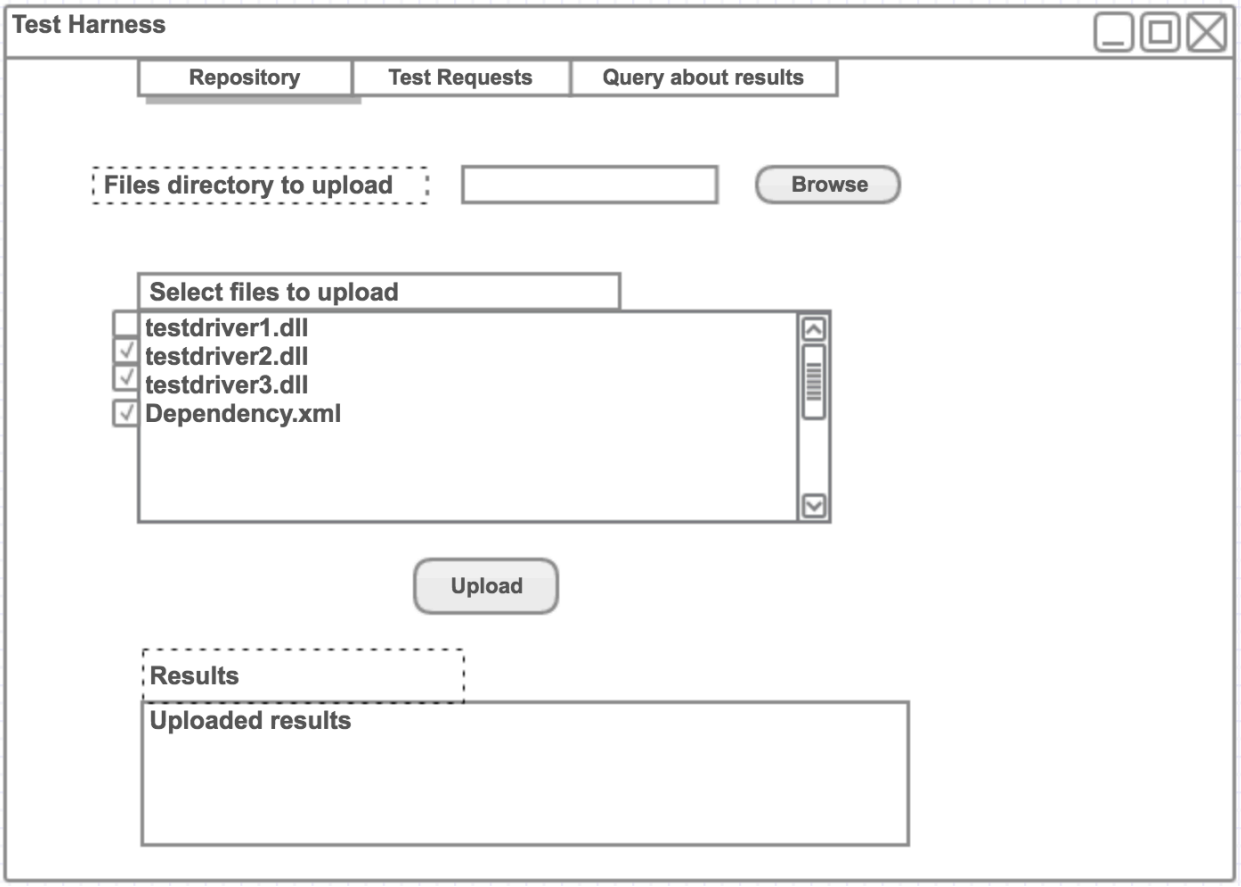


Figure 1

4.1.2 Test Request tab

This tab is providing all the testing related functionalities. Mainly, used by Developers and Quality Assurance team.

Developers or QAs who want to perform testing have to provide test driver name, and click on Add button to add test driver to "list of test drivers added". Up on clicking Submit button test request will be sent to Test Harness server.

Testing results will get displayed in Results section.

Figure 2 show Test Request tab features.

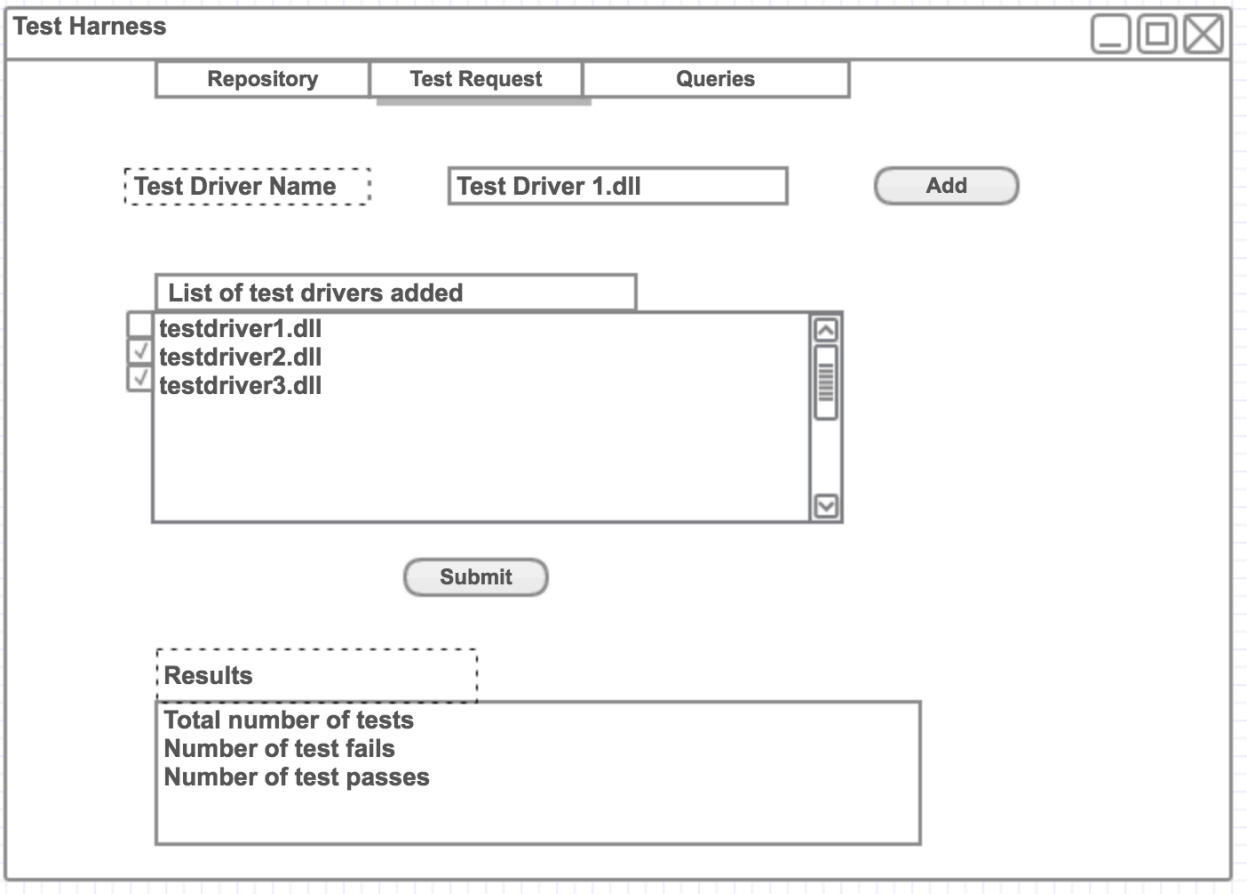


Figure 2

New features to implement

a. Providing testing whole project option for QA team, so that by clicking on single click whole application will get tested.
b. Test Request file uploading option, so that QA team can able to write list of test drivers in file and upload to test harness. Test Harness parse that file and starts testing.

4.1.3 Query tab

Query tab is primary for Project Managers, where this tab facilitates Project Manager to query about test results history. User Interface is providing lot of options like picking date range, request type, request for, selecting developer option.
Pick date option – User can select from date to end date to get test results based on date.
Request type – Two types of requests are present one is Simple results and other is Detailed results.
When user selects simple results only brief information about test results are displayed.
When user selects detailed results, detailed information about test results are displayed.
Sample simple results:
Author: Santhosh
Date: 09/18/2016
Test Name: Test Driver 1
Test Result: Success
Detailed Information results:
Author: Santhosh

Date: 09/18/2016
Test Result: Failure
Logs: failure due to file not found exception

Request for – User can select all results or he can select particular developers results by selecting developer's name.

Results will be displayed under Results section. Figure 3 shows Queries tab features.

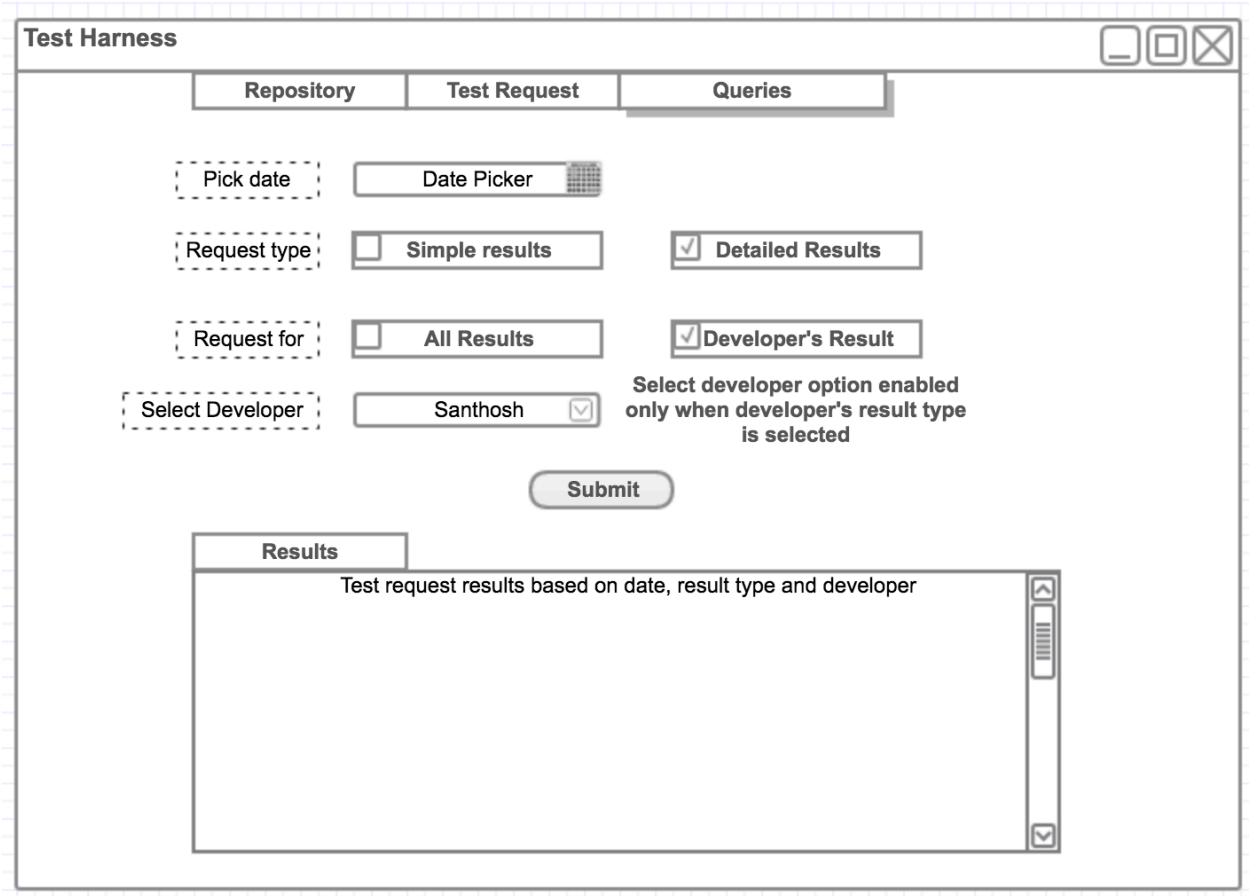


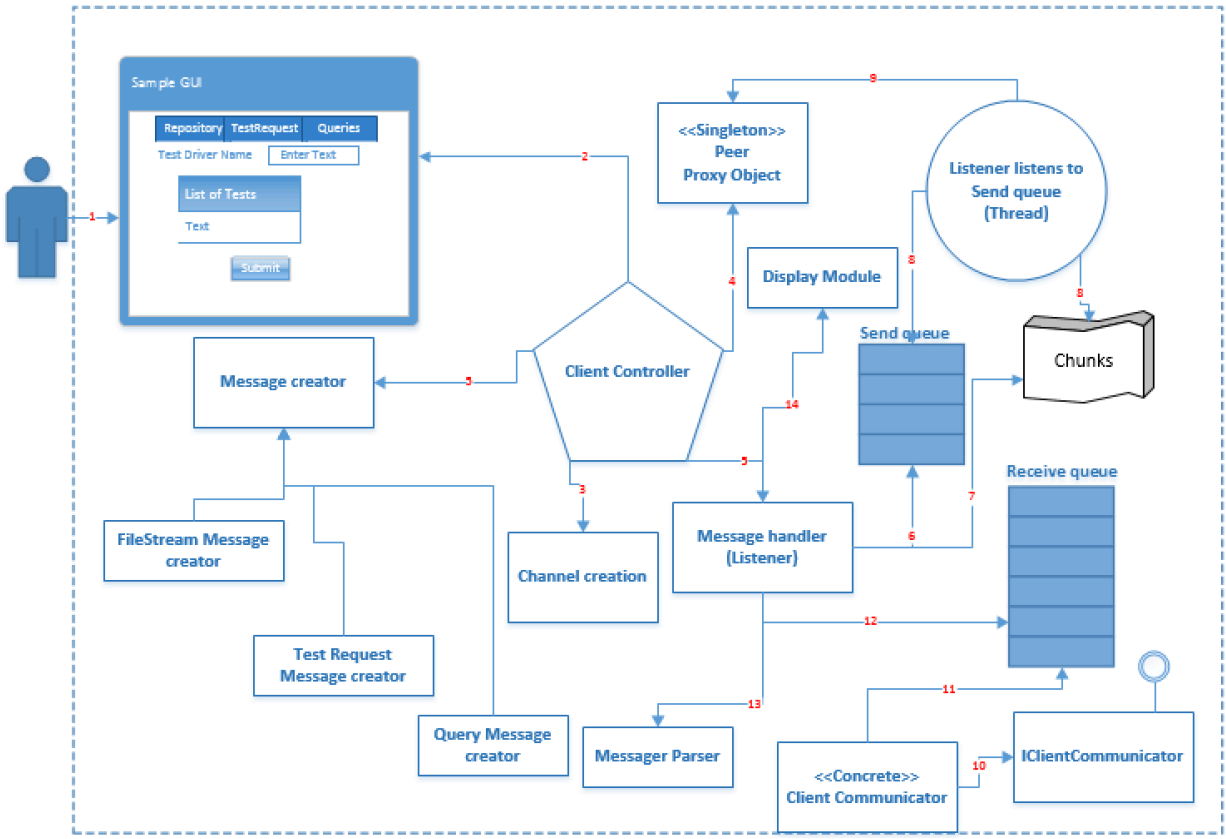
Figure 3

4.2 Test Harness Architecture

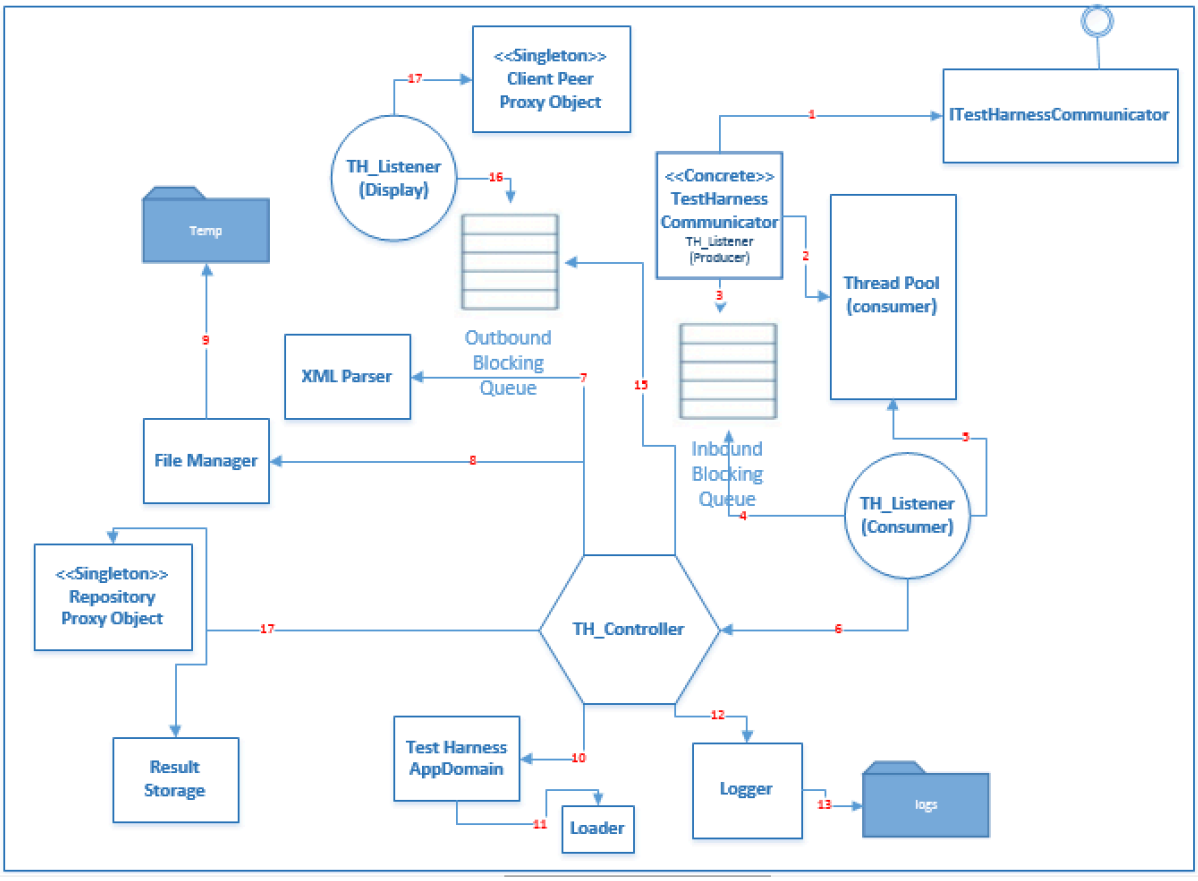
4.2.1 Client Application Architecture

1. User interacts with Test Harness through GUI through GUI Module.
2. Client controller acts as controller for whole client application. Main functionality of controller is to delegate calls from one module to other.
3. Client controller creates WCF channel, which helps in communication with test harness server.
4. Creates required proxy objects.
5. Based on user input client controller create message request in XML format. If client is querying, query message is created, similarly if client is asking for file upload request then file stream message is created.

6. With the help of message handler, controller will enqueues incoming requests to “send blocking queue”.
7. For file upload request, file chunks are created by client controller with help of file manager.
8. 9. Listener listens to “send queue” and for “file chunks”, and identifies the type of message and finally post that to server by using proxy object.
10. 11. If server wants to communicate with client, then server creates client proxy object and calls client communicator methods. All the results will be en-queued in client receive queue
13. 14.Message Handler listener listens to receive queue. By de-queueing it will parse the message using message parser. And send back to display module in client.



4.2.2 Test Harness Server



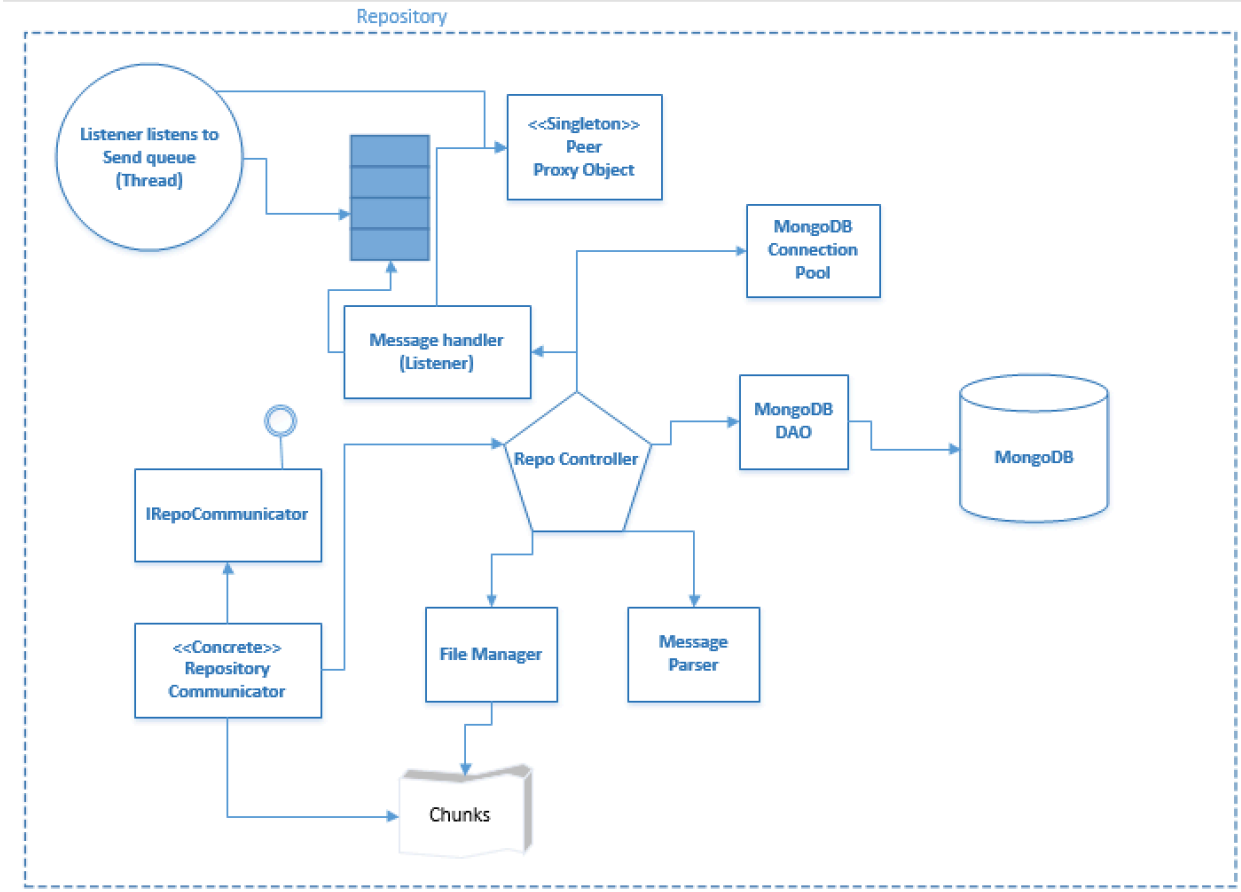
1. User requests Test Harness server to perform testing by posting test request to server using proxy object.
2. Multi Thread mechanism in Test Harness is implemented by adapting **Producer-Consumer Thread Pool design pattern** where producer acts as enqueuer for incoming requests to inbound blocking 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 application

   TH Controller will act as controller; it will call required component in respective packages to get work done.

7. Controller make use of XML parser to parse test request messages.
8. Controller make use of utilities in File Manager package to perform file related operations File Manager retrieves test drivers, application to be tested and its dependencies from Repository Server by using repository proxy object.
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 make use of Logger utility to log all the testing results in test harness's log directory.
13. Controller will store all the test results in repository server's MongoDB using proxy object created by result storage module.
14. Controller also inserts all test results to outbound queue
15. TH_Listener (Display Listener) keeps on listening to outbound queue if it finds any item in queue, then immediately it de-queues the item.
16. And listener respond to particular client by sending items present in outbound queue by using client proxy object.

4.2.3 Repository Server

Repository server will be used by both client and test harness server.
One of the main functionality of test harness application is a user can upload test drivers to repository server. To do this for client needs to send a message with location and filename to repository server. Repository server parses that message, creates proxy object and tries to pull the file form client. The main advantage we will get when server pull files from client is to check file existence in repository server.
Test Harness uses this repository server to download files that are required to process the test request. And it also uses to store test results in MongoDB.



4.3 Module or Package diagrams and interactions

The Complete Test Harness application is functionally separated and are designed in four layers. They are Test Harness layer, Repository layer, Client layer, Common layer. To make it simple, Common layer is not displayed in OCD. Common layer is nothing but modules that are common to all the other layers that are in place. For instance, Blocking Queue module is used in all the three layers so it can be in Common layer. As of now there is Blocking queue module that is in common. So it will be confusing when it is called as common layer instead of Blocking Queue. That's the reason why common layer is not discussed. But it is considered as assumption. But while implementing this project common layer will present.

4.3.1 Client Layer

Following are important modules used to develop test harness client application. Peers refers to Test Harness, Repository Servers (packages).

a. GUI Module:

GUI Module is responsible for having all possible screens that user can use to interact with Test Harness Application. User Interface is developed by using "Windows Presentation Foundation", one of the .Net feature. UI is designed in tabbed fashion which helps user to

easily navigate through all the features that Test Harness is providing. User Interface consists of following three tabs.

- Repository tab
- Test Request tab
- Queries tab

For more information about each and every tab refer User Interface section.

Interaction with other modules:

As mentioned in organizing principles, Application is designed by adapting MVC design pattern. Here GUI Module acts as View in Model-View-Controller design pattern. Where GUI interacts with Client Controller Module by sending information given by users. Client Controller Module will accept all the information and create message request, connection with other peer and process the request. Once Client Controller receives results, it will send back results to GUI Module for displaying results.

b. Test Executive:

Test Executive is giving console interface for Test Harness client application. This module helps in demonstrating all the requirements in project #4. And also act as one of the client.

Interaction with other modules: This module will be able to accept input form user and delegates user information to Client Controller module for further processing.

c. Client Controller Module:

Client Controller Module acts as controller for whole client application. Responsible for delegating information from one module to other, so that dependencies between modules will get reduced.

Interaction with other modules:

Client Controller module uses GUI module, Test Executive module, Peer connection module, blocking queue, File Management module, Message Module, Message Handling listener module.

d. Peer Connection Module:

Peer Connection Module is used to create channel between peers and to create proxy objects for particular services.

Interaction with other modules:

This module is called by Client Controller Module to create channel, proxy object if not exists.

e. Message Module:

Message Module is responsible to create messages for user requests in XML format and parse XML messages and convert to application readable format.

Interaction with other modules:

Message Module will be called by Client Controller Module to create messages for user inputs and is called by Message Handling Listener Module to parse and convert XML messages to application readable format.

f. Blocking Queue:

Blocking Queue is thread safe queue which is used for communicating messages with other peers such as Test Harness, Repository peers.

Interaction with other modules:
Blocking queue module interacts with Client Controller module, Message Handling Listener module and TH Client Services. Client Controller module enqueues messages received from Message module (XML Format messages). Message Handling Listener keeps on listening blocking queue and sends messages to respective peer. TH Client services keeps on queuing result messages coming from other peer to blocking queue.

g. HiResTimer Module:
High Resolution Timer Module is used to get latency between calls.
Interaction with other Modules:
This Module is mainly used by Test Executive to display latency time for each and every calls while demonstrating requirements.

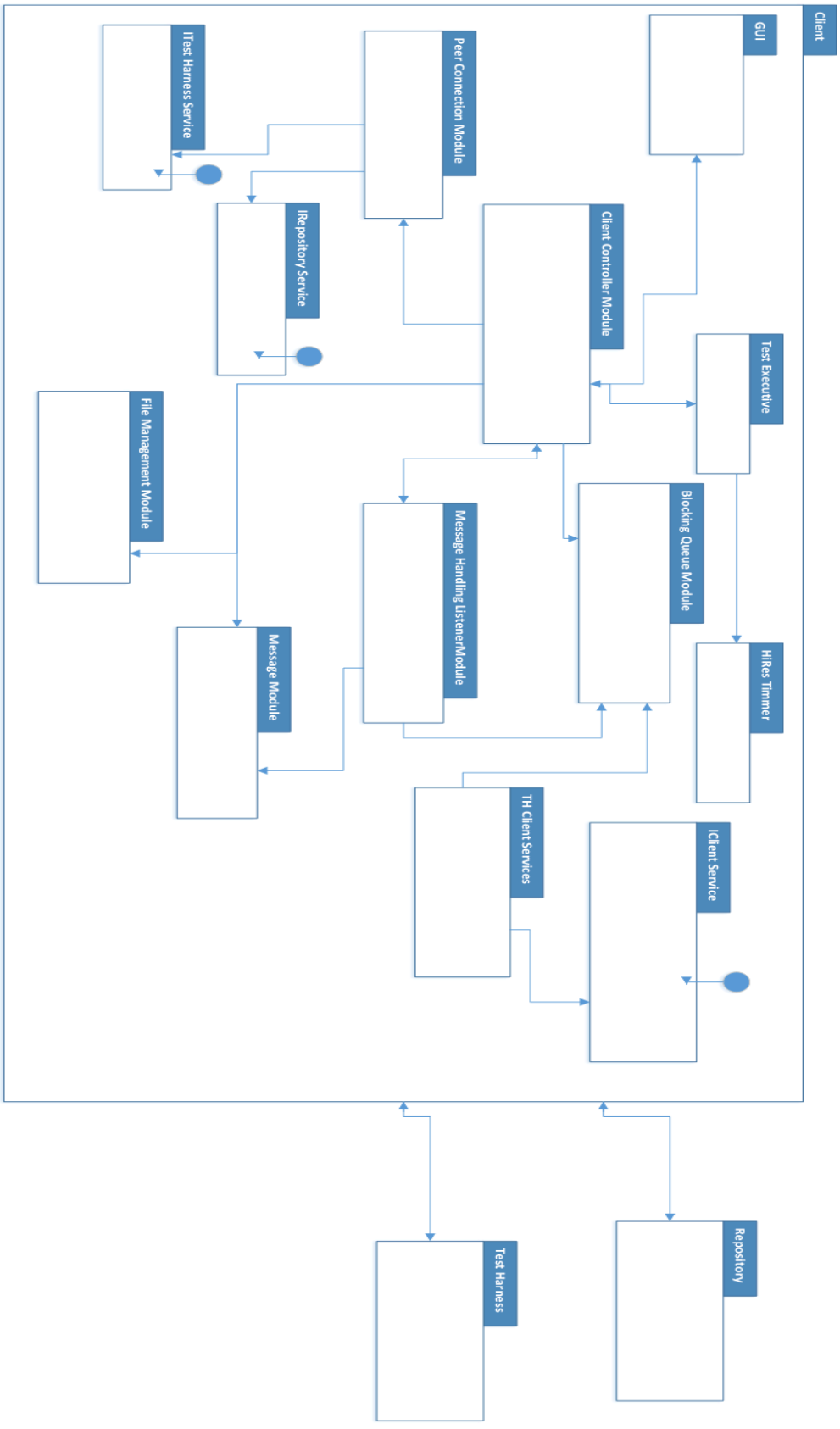
h. IRepository Service, ITestHarness Service, IClient Service Module
IRepository, ITestHarness, IClient Services provides service contract for communication between peers. This contract tells which services methods are exposed over WCF channel. If service contract method is not declaring as operation contract that method is not exposed to peers.
Interaction with other modules:
This services are mainly used by peer connection module which is responsible to create channel and proxy objects.

i. File Management Module
File Manager comes under common layer. Following are functionalities provided by File Manager

- Displays all the file present in specified directory given by user from GUI.
- Responsible for pulling up files from other peers and downloading files, these two are done in the forms of chunks.
- Creating directories if needed.

j. Message Handling Listener Modules
Message Handling Listener module plays an important role in communication between peers. This module keeps on listening to blocking queue module if any message is enqueued in blocking queue this module dequeues that message and starts processing. If message is from sender blocking queue this module identifies what type of message and send to particular peer machine. If message is from receiver blocking queue this module parse message and displayed in console and GUI.
Interaction with other Modules:
This module is initiated by controller module and keeps on listening to blocking queue module.

k. TH Client Services Module
This module implements IClientService service contract.
Interaction with other Modules:
This module is interacting with blocking queue module, when any peer requesting its methods this module enqueues those requests in blocking queue.


Client
GUI
Peer Connection Module
ITest Harness Service
IRepository Service
Client Controller Module
Test Executive
File Management Module
Blocking Queue Module
Message Handling ListenerModule
Message Module
HiRes Timmer
TH Client Services
IClient Service
Test Harness
Repository

4.3.2 Test Harness Layer

Following are important modules used to develop Test Harness application.

a. Test Harness Service

Test harness service module implements ITestHarness service contract module. Responsible to accept all the incoming test requests from clients, Initiating thread pool, blocking queue.
Interaction with other modules:
This module interacts with blocking queue by en-queuing the incoming request, Thread pool, Test Harness controller module.

b. Thread Pool

When application is started this module creates required number of threads to process requests concurrently. Advantage of using this module:

1. Thread creation time will get reduced
2. Thread management will become easy.

Thread pool Design: Create required number of threads and store all those in an array (should be done in static block). A "Start All Threads" function will start all threads.
Create a blocking queue such that all threads listen to that queue. When there are no messages present in blocking queue, threads go to wait state. When message is enqueued in blocking queue, it will notify the threads. One of the thread picks up that message and start processing the message.
Interaction with other module:
Thread pool uses blocking queue, test harness controller module.

c. Message Handling Listener

Message Handling Listener module plays an important role in communication between peers. This module keeps on listening to blocking queue module if any message is enqueued in blocking queue this module dequeues that message and starts processing. If message is from sender blocking queue this module identifies the type of message and send to particular peer machine.
Interaction with other modules:
This module is initiated by controller module and keeps on listening to blocking queue module, and finally interact with peer connection module.

d. Blocking Queue Module

Blocking Queue is thread safe queue which is used for communicating messages with other peers such as Client, Repository peers.
Interaction with other modules:
Blocking queue is used by test harness controller module and thread pool.

e. Application Domain Module

Application Domain module creates child application domains of each test requests. And also used to create child application domain proxy object. Following are some advantages we will get by running DLL files in AppDomain.

- AppDomain creates isolation process for each DLL so that it will continue processing even when there is failure in code execution.

- Child processes are created with secure environment by AppDomain.

Interaction with other modules:
Application domain module will get called by Test Harness controller to create child application domain.

f. Loader Module
This module loads all the execution libraries required for the test request and also it starts running test drivers with help of reflection feature provided by C#. Net
Interaction with other modules:
This module interacts with Test Result modules which contains serialization class to store test results.

g. Logger Module
This module is used to log all the testing related messages in log files.
Interaction with other modules:
This module is used by Controller module, which sends messages to log in files.

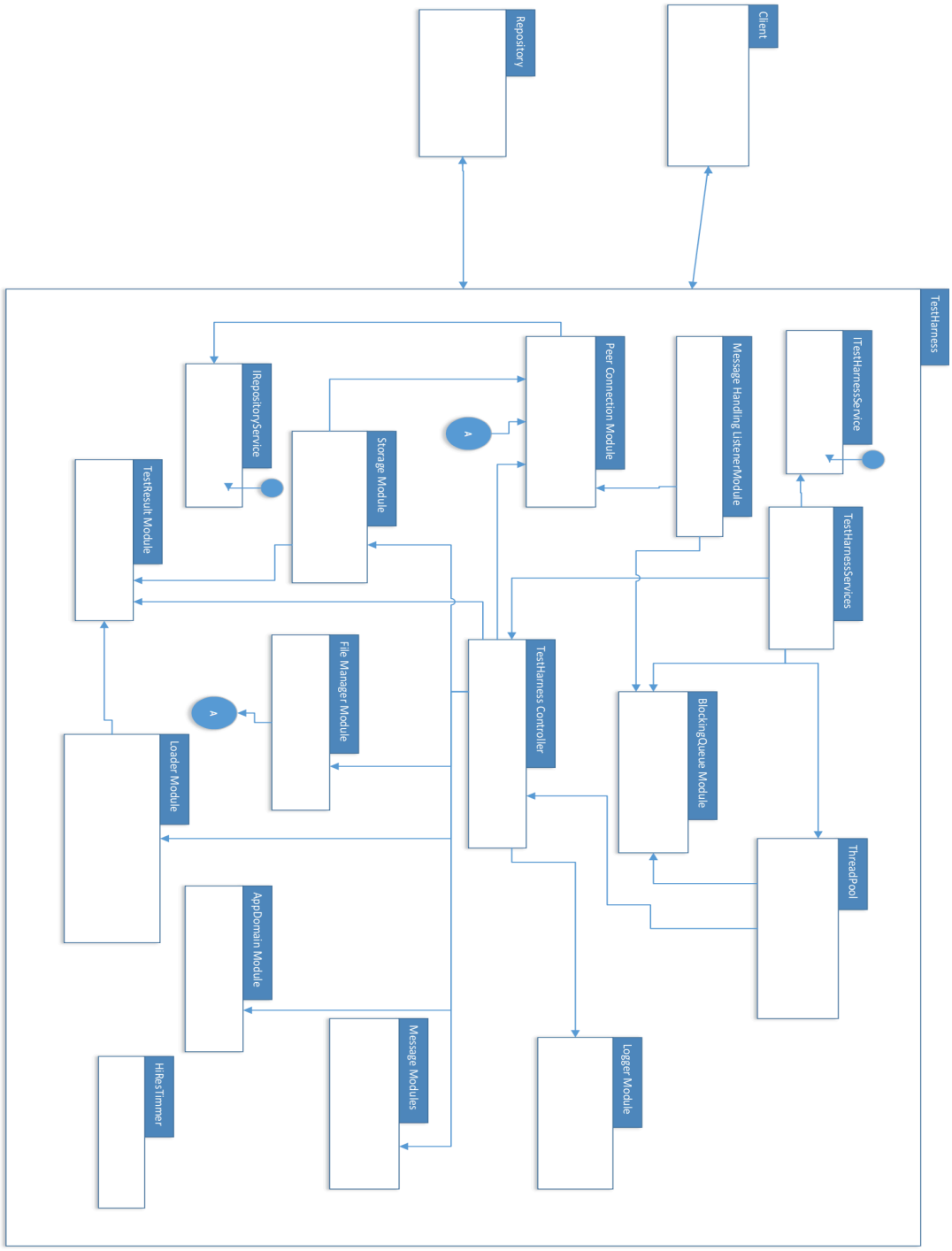
h. Peer Connection Module
Peer Connection Module is used to create channel between peers and to create proxy objects for particular services.

i. File Manager Module
File Manager comes under common layer. Following are functionalities provided by File Manager

- Displays all the file present in specified directory given by user from GUI.
- Responsible for pulling up files from other peers, downloading files, these two are done in the forms of chunks.
- Creating directories if needed.

j. Message Module
Message Module is responsible to create messages for user requests in XML format and parse XML messages and convert to application readable format.
Interaction with other modules:
Message module is used by test harness controller, which helps in creating message, parse message.

k. IRepository Service, ITestHarness Service, IClient Service Module
IRepository, ITestHarness, IClient Services provides service contract for communication between peers. This contract tells which services methods are exposed over WCF channel. If service contract method is not declaring as operation contract, then that method is not exposed to peers.



4.3.3 Repository Layer

1. MongoDB Connection Pool Module

This module is used to maintain MongoDB connection instances. Connection pool is nothing but maintaining list of connection instances as like thread pool. Following are some advantages of using connection pool.

a. Minimize creation of new connection instances
b. Effective usage of connection objects
c. Reduce of number of stale connection objects

2. DAO Module

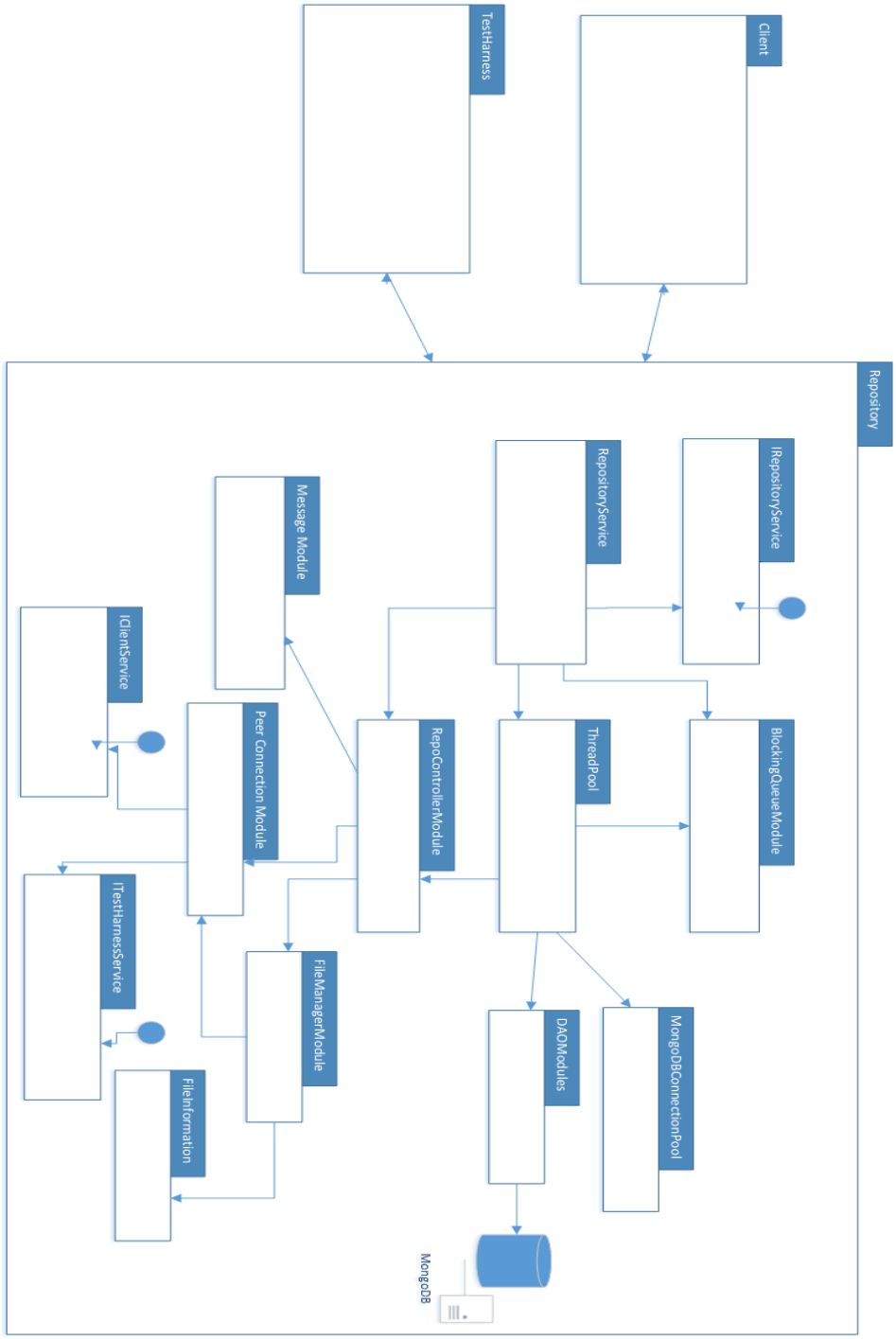
This module consists of two Data access object classes; one is for Logging test requests and other is for storing dependency information about files. Since we are using MongoDB as storage, the data to insert should be in JSON format.

Following is sample test request results in JSON format.


```
{
    "author": "Piduri Santhosh",
    "listOfTestDriverResult": [{
            "listOfTestCaseResults": [{
                    "logs": "\u000a entering into TestDriver5 - > test() method\u000a divides
by zero but does not try to catch exception\u000a Result :- False",
                    "testName": "TestDriver.TestDriver5",
                    "testResult": "False"
            }, {
                    "logs": "\u000aResult :- True",
                    "testName": "TestDriver.TestDriver11",
                    "testResult": "True"
            }, {
                    "logs": "\u000a entering into TestDriver6 - > test() method\u000a calls
thread.abort() and tries to catch exception",
                    "testName": "TestDriver.TestDriver6",
                    "testResult": "False"
            }],
            "testDriverName": "TestDriver2, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=null"
    }],
    "testRequestName": "Piduri Santhosh131201970635793646",
    "timeStamp": "2016-10-05 23:11:03"
}
```


3. File Information Module

This module holds file information such as file name, creation date, file size etc.



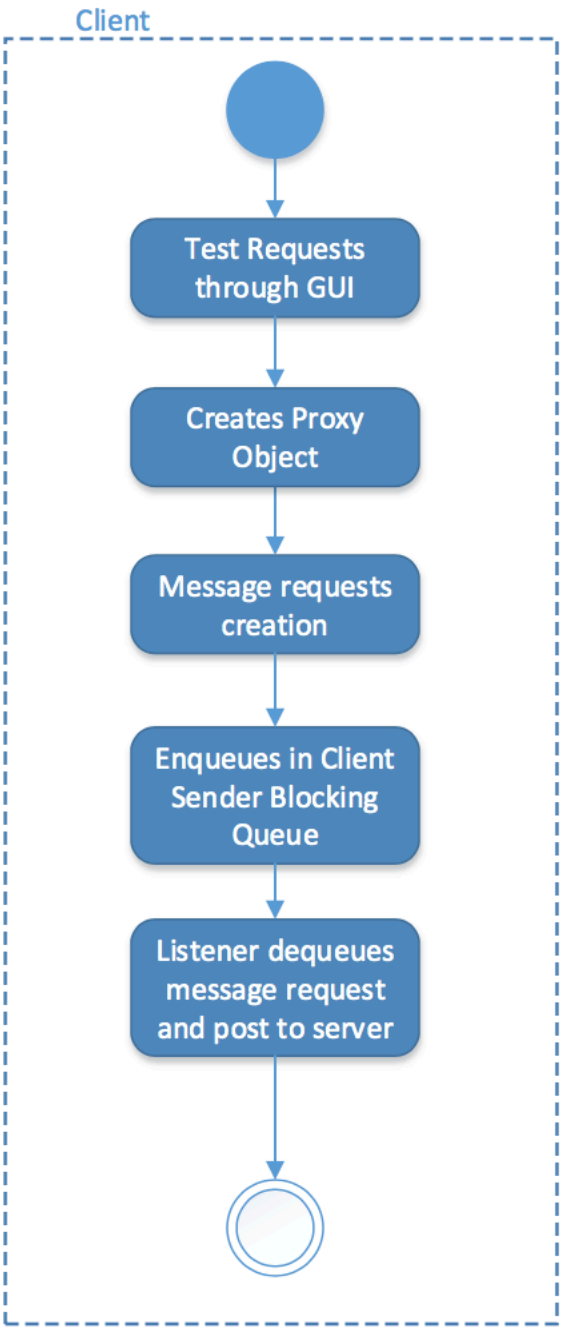
4.4 Activity Diagrams of Test Harness

4.4.1 Overall system activity diagram about Test Request

Following activity diagram describes overall system. This diagram is divided into two partitions one is client and other is server.

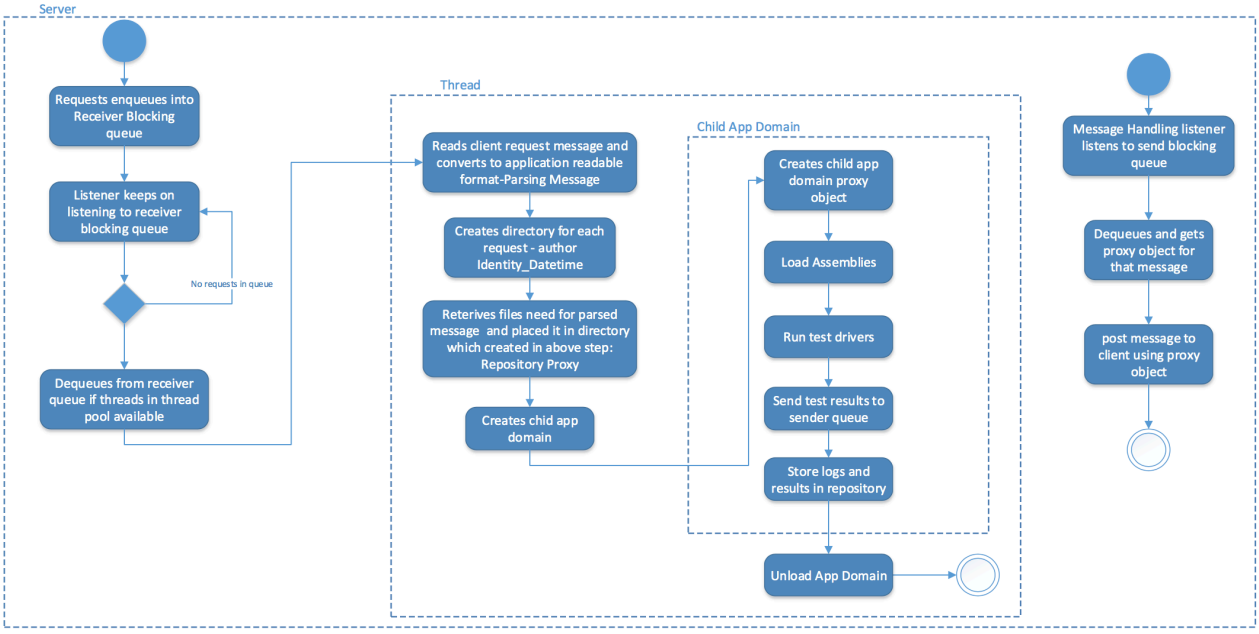
Following steps describes client activity diagram.

1. User interacts with test harness by using GUI which is provided by test harness client application. First user has to click on Test Request tab, to make test request. Test request needs List of test driver names. Once provided he has to click on Submit button.
2. Application connects to test harness server and converts test request into XML format using Message module.
3. Enqueues converted messages into sender blocking queue.
4. Message handling listener dequeues message from queue, with the help of peer object Listener and posts message to Test Harness server.



Following steps describes server activities.

1. All the client requests are enqueued into receiver blocking queue by Test Harness Service module.
2. Message handling listener dequeues each request and allocate a thread from thread pool. If threads are busy in thread pool listener won't dequeue requests and it waits for a thread.
3. Message parser is used to parse request messages and convert to application readable format.
4. Creates separate directory inside application base path for each test requests. Directory name will be author followed by timestamp.
5. File Manager download files that are required for a request into directory that was created. These files are downloaded from repository server using proxy object.
6. Controller creates child application domain to perform testing.
7. Creates child application domain proxy object.
8. Child application domain load all assemblies required for the request.
9. Run test drivers using reflection concepts.
10. Child application domain send results to sender blocking queue, so message handling listener dequeues and sends to respective clients
11. Finally, all the test results are stored in MongoDB in repository server and logs are logged in Test Harness server.
12. Main application domain will unload child application domain and deletes the directory which is used by child application domain.



How Test Harness server sends request results to client?

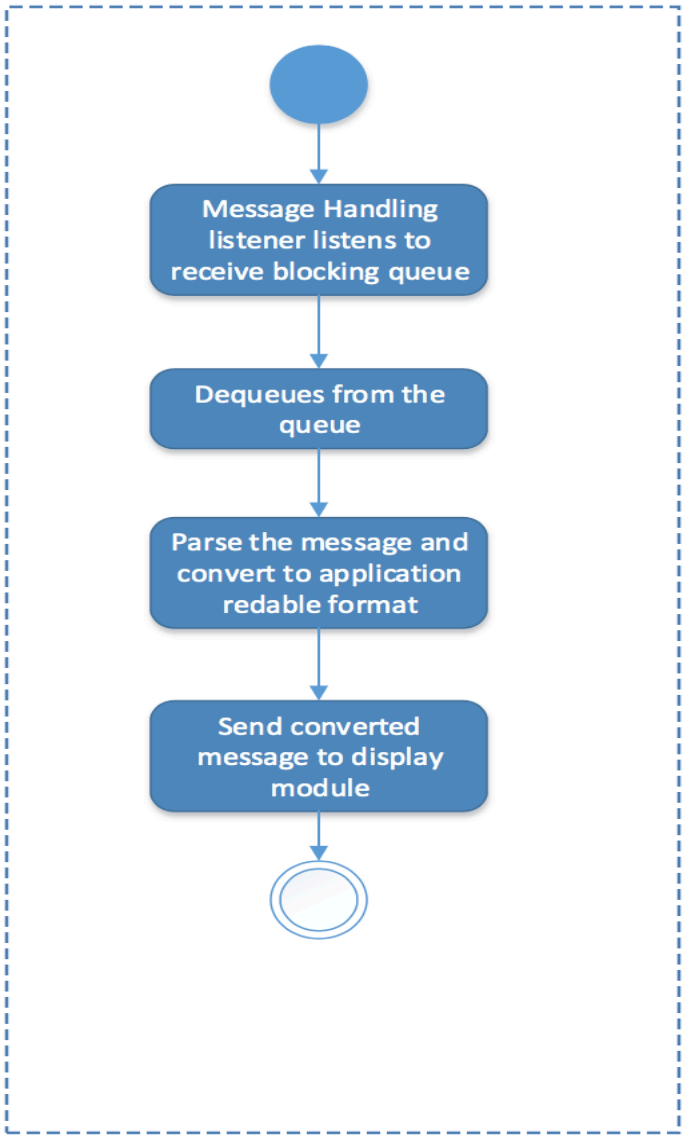
Following steps describes sending messages to client from test harness.

1. Message Handling listener listens to send blocking queue.
2. Listener dequeues the messages and get proxy object for that message.
3. Test Harness posts message to client using that proxy object.

How Client accepts and display results coming from other peers?

1. Message Handling listener listens to receive blocking queue in client.
2. Dequeues from the queue.

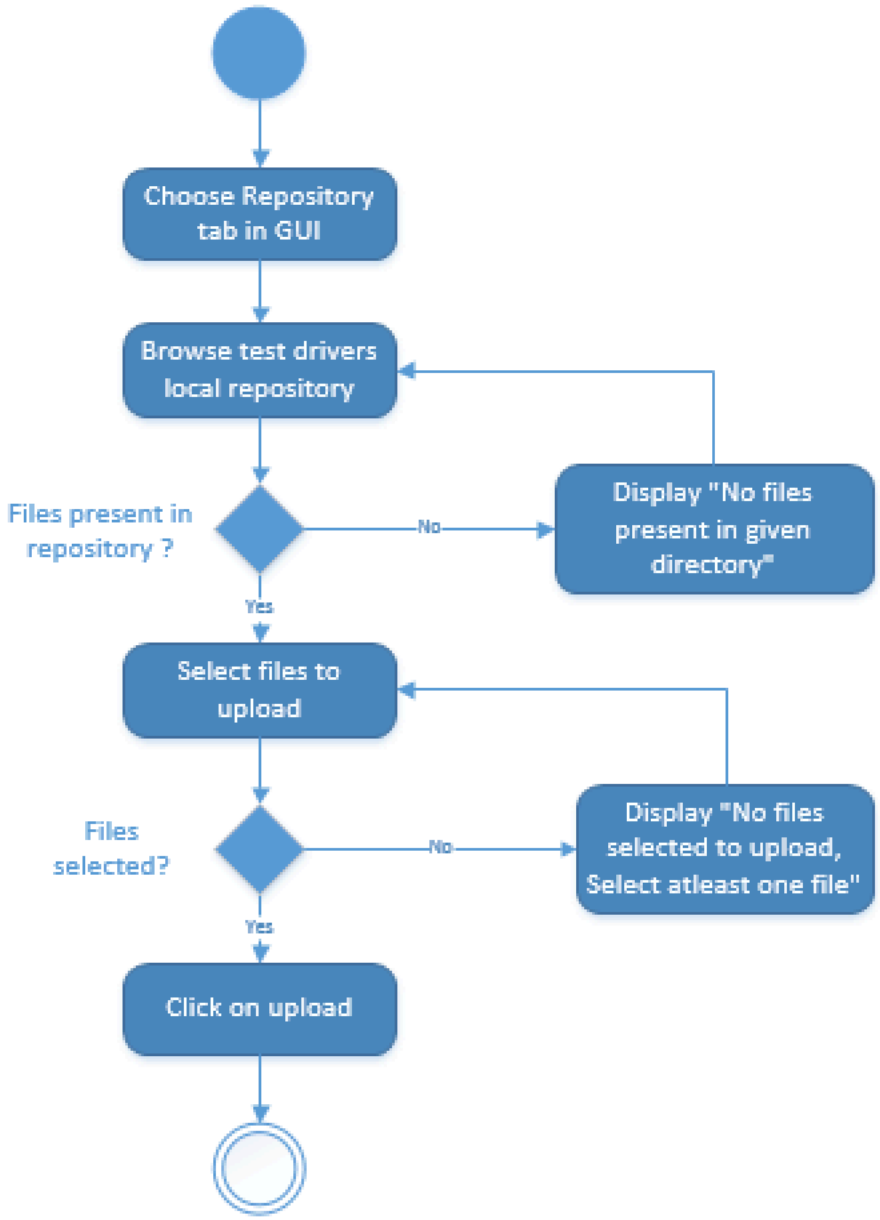
3. Parse the message and convert to application readable format.
4. Sends converted message to client display module.



4.4.2 Repository Activity diagram

Following steps describes file uploading part in repository tab to repository server.

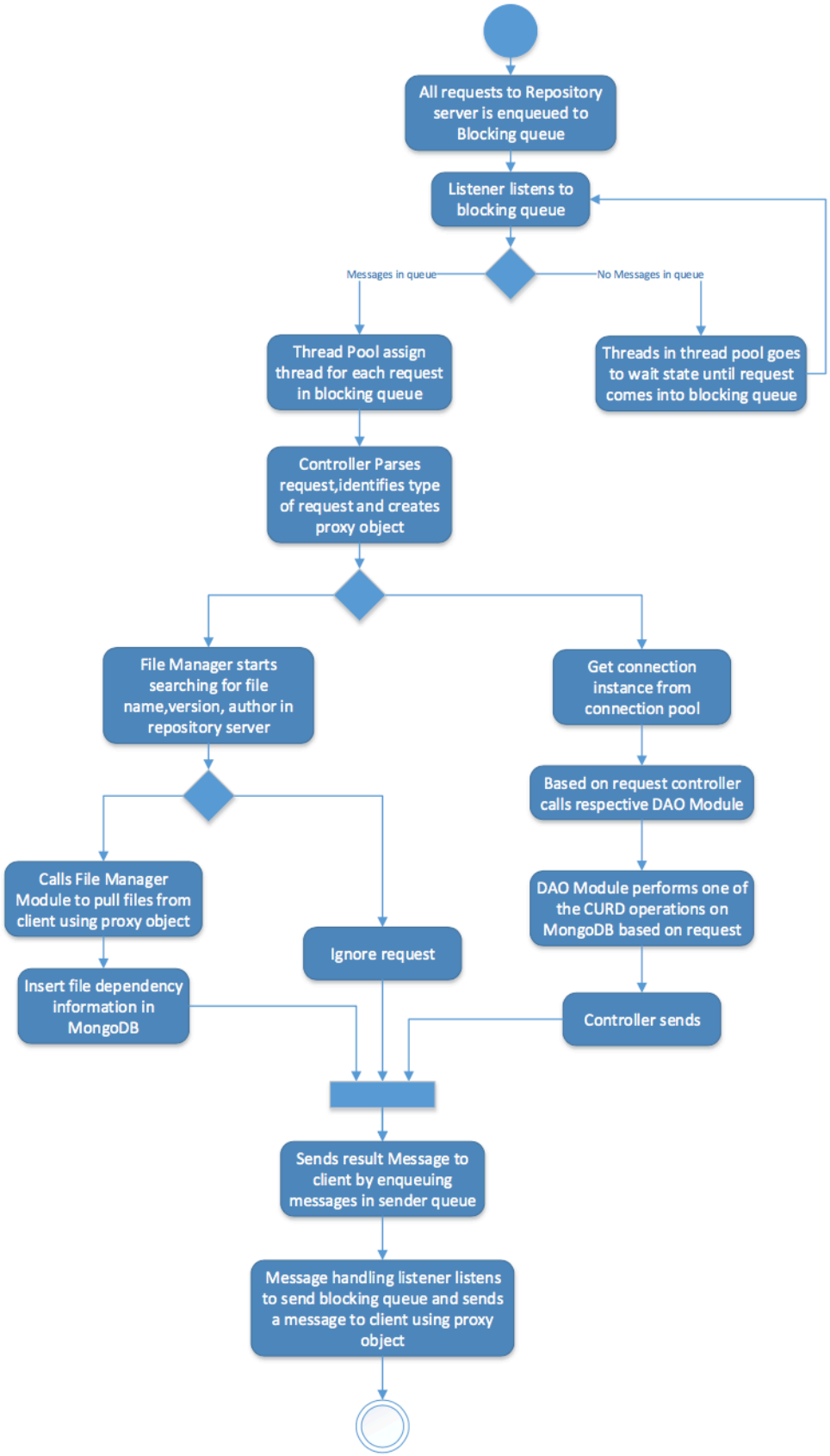
1. User has to select Repository tab in GUI
2. Browse test driver's location (directory path).
3. Checking if files are present in the directory, if not display messages as "No Files present in the given directory". If yes goes to next step.
4. User should select at least one files to upload.



5. If no files are selected, then display message as “No Files selected to upload, Select at least on file”.
6. Click on upload button to start uploading files.

Following activity diagram for operation in repository server side.

1. After clicking upload button (refer above flow), All requests are en-queued to receiver blocking queue of repository server with the help of Repository service module.
2. Message Handling listener listens to blocking queue.
3. If messages present in queue, Thread Pool allocate thread for each message in blocking queue.
4. If no messages present in queue, then all threads in thread pool goes to sleep or wait state.

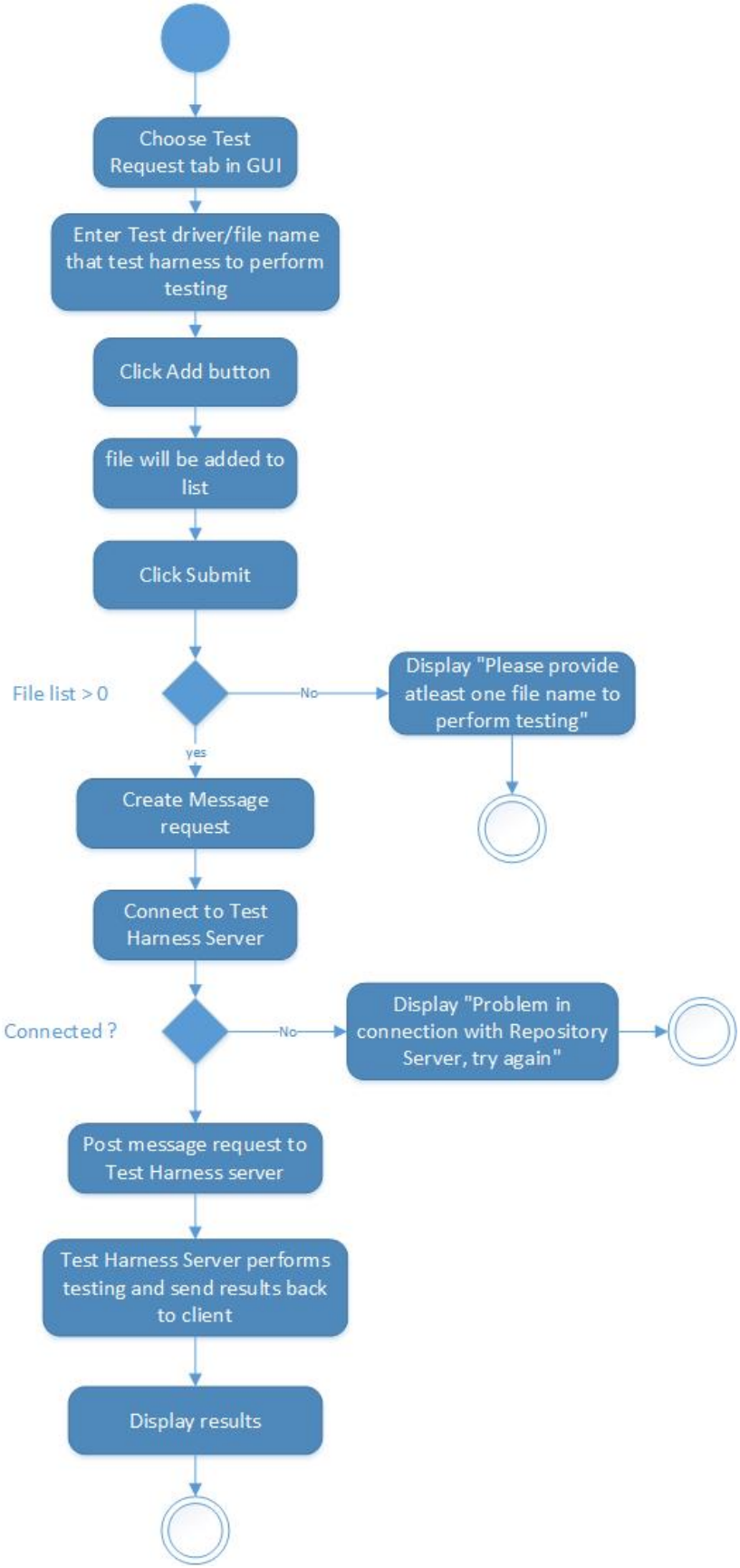

All requests to Repository server is enqueued to Blocking queue
Listener listens to blocking queue
Messages in queue
No Messages in queue
Thread Pool assign thread for each request in blocking queue
Threads in thread pool goes to wait state until request comes into blocking queue
Controller Parses request,identifies type of request and creates proxy object
File Manager starts searching for file name,version, author in repository server
Get connection instance from connection pool
Based on request controller calls respective DAO Module
Calls File Manager Module to pull files from client using proxy object
Ignore request
DAO Module performs one of the CURD operations on MongoDB based on request
Insert file dependency information in MongoDB
Controller sends
Sends result Message to client by enqueuing messages in sender queue
Message handling listener listens to send blocking queue and sends a message to client using proxy object

5. Controller parses messages using message module to identify type of message.
6. If message is related to file then File Manage module checks for file existence by considering file name, version, author.
7. If file is new, then file manager module gets client proxy object and pulls the file.
8. Also inserts file dependencies in MongoDB using DAO Modules
9. If file exists, then repository server just ignores the request and sends back message to client saying that file already exists.
10. If message is related to Database related operation, then controller will get its connection object from connection pool.
11. Controller will call respective DAO Module based on the request.
12. DAO Module performs one of the CURD operations on MongoDB. CURD refers to create, update, retrieve, delete operations.
13. Controller sends message to sender blocking queue.
14. All the results will be enqueued in sender blocking queue.
15. Message handling listener listens to sender blocking queue and then dequeues and finally sends that to client by using client proxy object.

4.4.3 Test Request Activity diagram

Following steps describes Test Request activity diagram.

1. User should select Test Request tab to perform test request.
2. User should provide name of the test driver on which test harness should perform testing
3. User should click on add button to add test driver name to list. This steps repeats
4. All files will be added to file list.
5. Click on submit button.
6. Client application converts request into xml format
7. Connects to Test Harness server.
8. Posts request to Test Harness server by using proxy object.
9. Test Harness starts processing this request and sends back the results to client.


Choose Test Request tab in GUI
Enter Test driver/file name that test harness to perform testing
Click Add button
file will be added to list
Click Submit
File list > 0
No
Display "Please provide atleast one file name to perform testing"
yes
Create Message request
Connect to Test Harness Server
Connected ?
No
Display "Problem in connection with Repository Server, try again"
Post message request to Test Harness server
Test Harness Server performs testing and send results back to client
Display results

5 Critical Issues

1. Issue #1
   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.

2. Issue #2
   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.

3. Issue #3
   Security: In Test Harness, Developers will upload project files into the repository server to perform testing. How data integrity of those files will be maintained?
   Solution: To maintain data integrity of files, there should some security while sending files from client to server. To achieve this secure communication between client and server, we are implementing WSHTTP binding provided by "Windows Communication Foundation". By using this feature, files will get encrypted before sending from client to server.

4. Issue #4
   Test Harness has to support multiple client requests; Application creates multiple threads in order to process client requests. Though CLR is providing thread pool, it will be difficult to know how many threads are running at a time and maintaining them. Most of the times Test Harness will be very busy with many client requests, if we allow application to create threads by itself using CLR there will performance issues. For instance, If Test Harness take 5 seconds to process a simple request, among those 5 seconds application will take 3secs to create thread. So how can we easily keep track, maintain threads and increase performance of Test Harness considering the above situation?
   Solution: Test Harness also should maintain thread pool. When an application starts, it will also start a set of threads and keep it in a pool. Threads in a thread pool will be waiting or sleeping until requests comes to blocking queue. Listener notifies threads when a request

is enqueued in the blocking queue. By maintaining thread pool in this application, thread creation time will be reduced, which improves performance of an application.
Thread Pool code will be as follows

```
public THConsumerThreadPool(int noOfThreads)
        {

           //this.blockingQueue = blockingQueue;
           threads = new ThreadPool[noOfThreads];
           for (int i = 0; i < threads.Length; i++)
           {
              threads[i] = new ThreadPool();//blockingQueue);
              ThreadStart threadStart = threads[i].threadFunc;
              Thread thread = new Thread(threadStart);
              thread.Start();

           }
    }

public void threadFunc()
        {
           while (true)
           {

              lock (locker_)
              {
                 while (blockingQueue.size() == 0)
                 {
                    Console.Write("\n waiting");
                    Monitor.Wait(locker_);
                 }
                 Action act = blockingQueue.deQ();
                 if (act != null)
                 {
                    Console.Write("\n Thread" + Thread.CurrentThread.ManagedThreadId);
                    act.Invoke();
                 }
              }
             // Thread.Sleep(5000);
           }
        }
```

5. Issue #5
   Developers who are following this OCD are supposed to create and parse message requests in XML format. How would developers know the various xml tags that can be used to create and parse message requests?

Solution: Message designers will provide XSD file to all developers. 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

a. During compile time itself user can able to figure out what is wrong with his request file.
b. 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>
```

6. Issue#6
   How Test Harness application handles when multiple requests want to make use of same DLL file?
   Solution: For each test request, application will create and load all files in a directory which is particularly created for that request. And then child application domain will run tests on the files in that directory. In this way application will maintain a separate copy of DLL files for each requests. By this way multiple requests that are referring to the same DLL file can be handled.

6 References

1. My Project#1 Operational Concept Document.
2. Dr. Fawcett code repository.

7 Appendix

7.1 Message creation and Parsing prototype

Message transfer between peers will be in format of XML. Sample message request will be as follows.

```
<TestHarness>
 <MetaInfo>
  <ServerAddress>192.13.124.1</ServerAddress>
  <ClientAddress>127.0.0.1</ClientAddress>
  <MessageType>TestRequest</MessageType>
 </MetaInfo>
 <Body>
  <Test name="Test1">
   <TestDriver>HelloTestDriver</TestDriver>
   <Author>santhosh</Author>
   <Library>CodeToTest1</Library>
   <Library>COdeToTest2</Library>
  </Test>
 </Body>
</TestHarness>
```

Whole message is divided into two parts; one is meta info and other is body. Meta Information contains information about server address, client address and message type. Body contains information about Tests to be performed.
Prototype code uses XML Link .Net feature to create and parse messages.
Following code creates above message elements.

```
xml = new XDocument();
```

```
 xml.Declaration = new XDeclaration("1.0", "utf - 8", "yes");
 XComment comment = new XComment("This is messag request");
 XElement root = new XElement(rootNode);
 xml.Add(root);
 XElement metaNode = new XElement(metaInfoNode);
  xml.Root.Add(metaNode);
 XElement serverAddress = new XElement(metaInfoNode_serverAddress,
 metaInformation.serverAddress);
 XElement ClientAddress = new XElement(metaInfoNode_ClientAddress,
 metaInformation.clientAddress);
 XElement MessageType = new XElement(metaInfoNode_MessageType,
 metaInformation.messageType);
 metaNode.Add(serverAddress);
 metaNode.Add(ClientAddress);
metaNode.Add(MessageType);
XElement bodyTag = new XElement(body);
xml.Root.Add(bodyTag);



int i = 0;
foreach(Test test in tests)
{
   i++;
   XElement testTag = new XElement("Test");
   testTag.SetAttributeValue("name", "Test" + i);
   XElement testDriver = new XElement("TestDriver", test.testDriverName);
   XElement author = new XElement("Author", test.author);

   testTag.Add(testDriver);
   testTag.Add(author);

   foreach (string library in test.libraries)
   {
      XElement Library = new XElement("Library", library);
      testTag.Add(Library);
   }
   bodyTag.Add(testTag);
}
```

Following code is used to parse metadata information from the above message request.

```
I        Enumerable<XElement> xtestCode =doc_.Descendants("MetaInfo").Elements();
   foreach (var xlibrary in xtestCode)
   {
      if(xlibrary.Name == "ServerAddress")
```

```
            {
               metaInfo.serverAddress = xlibrary.Value;
            }else if(xlibrary.Name == "ClientAddress")
            {
               metaInfo.clientAddress = xlibrary.Value;
            }else if (xlibrary.Name == "MessageType")
            {
               metaInfo.messageType = xlibrary.Value;
            }

         }
```

Prototype out will be as follows

```
<TestHarness>
  <MetaInfo>
    <ServerAddress>192.13.124.1</ServerAddress>
    <ClientAddress>127.0.0.1</ClientAddress>
    <MessageType>TestRequest</MessageType>
  </MetaInfo>
  <Body>
    <Test name="Test1">
      <TestDriver>HelloTestDriver</TestDriver>
      <Author>santhosh</Author>
      <Library>CodeToTest1</Library>
      <Library>COdeToTest2</Library>
    </Test>
  </Body>
</TestHarness>
===============Parsing meta data info ============
Server Address ==192.13.124.1
  Client Address ==127.0.0.1
  Message Type ==TestRequest
 Press any key to continue . . .
```

7.2 Message passing communication prototype

Prototype code uses Windows communication foundation .Net feature to communication between peers. Server has to create channel, which allows clients to communicate with each other. Following code shows how to create communication channel.

```
static ServiceHost CreateChannel(string url)
    {
       BasicHttpBinding binding = new BasicHttpBinding(); //uses basichttpbinding
       Uri address = new Uri(url);
       Type service = typeof(TestHarnessServices);
       ServiceHost host = new ServiceHost(service, address);
       host.AddServiceEndpoint(typeof(ITestHarnessServices), binding, address); //creates end point address.
       return host;
    }
```

Following code show how to create proxy for remote server.

```
static C CreateProxy<C>(string url)
      {
         BasicHttpBinding binding = new BasicHttpBinding();
         EndpointAddress address = new EndpointAddress(url);
         ChannelFactory<C> factory = new ChannelFactory<C>(binding, address);
         return factory.CreateChannel();
      }
```

If server wants to expose its services, server should provide service contract which intern describes which services are exposing. Following code shows how to create service contract

```
[ServiceContract]
   public interface ITestHarnessServices
   {
      [OperationContract]
      void processRequest(string testRequest);
   }
```