Test Harness

Operational Concept Document

CSE681-PROJECT #1

By You Tian (SUID: 437720484)

Instructor: Jim Fawcett

Date: 14th September, 2016

Contexts

Figures

1. Executive Summary

This document illustrates the operational concept of the Test Harness system.

Introduction:

Big system has become very popular in today’ s software industry. Big systems shall not be possibly developed by single developer. Developers, quality assurance engineers and project managers will always struggle with the issue that, how we can fit the modules and packages developed by different people into the baseline of big system. Test Harness is designed to help us solve this issue.

Uses:

Test Harness is designed to meet requirements from 5 types of users. Every user can choose the user mode in the command line when start the Test Harness.

Application Activities:

The first step shall be accepted test requests in the form of XML files. The XML files will be changed into string format and enqueue the queue. Then Test Harness create child domain for the XML file, and the child appdomain request the queue to dequeue new XML string. XML string shall be decoded into application readable format in child appdomain. File Manager package shall search the directory tree and find DLL files in the repository for further processing.

Then DLL files will be loaded in the child appdomain. Test Loader read the test driver, and load the test library. The test() method in ITest can be call by Test Executive, taking no arguments and returning the test pass status, e.g., a Boolean true or false value. The getLog() function in ITest can be used to retrieve logs from the child appdomain. Child appdomain will merge the results and logs according to users' mode, then it will query the Queue. If the XML queue is not empty, child appdomain will be deleted and new child appdomain will be created for next XML string. If the XML queue is empty, Test Harness will delete the child appdomain and end the test process.

Partition:

Test Harness will be implemented in C# using .NET framework on visual studio 2015. The system will be divide into 12 modules. Main modules are as below:

- Test Executive- holds the main method, and is responsible for the main control flow of the system.
- Appdomain Manager- is used to create child appdomain and call the Test Executive to do the test.
- ITest- contains test() method which can be call by Test Executive, taking no arguments and returning the test pass status, e.g., a Boolean true or false value

Critical Issues

In implementing Test Harness, several critical issues shall be carefully considered. Their possible solutions have been given, and some of the issues are as blow:

- Requests handling- in which Test Harness get test requests and change into application readable format.
- Performance- how to deal with the extreme situation, like hundreds of test requests flood in or single test request contains hundreds of DLL files.
- Exception handling- how to handle the exceptions occurred in both Test Harness and child appdomain.

The Test Harness thus ensures that it can handle the test requests, do the test successfully and demonstrate results clearly.

2. Introduction

With the remarkable advance in both computer hardware technology and software engineering technology, developers are able to create huge projects with complex structures to meet various demands.

When we develop large projects, we should always struggle with the issue that, how we can successfully insert the packages, classes and modules developed by different developers into the baseline of the final project. Test Harness is designed to make sure the code can fit into the baseline without errors.

1. Application Obligations

The primary requirement of Test Harness is testing the code. So that the code written by different developers can fit into the baseline without any error. The main obligations of the application should be:

- To accept one or more Test Requests, each in the form of an XML file.
- To create a child appdomain that isolates test processing from Test Harness processing, and delete it when test ends.
- To load the test library and execute the test() method which returns the test results and store them in repository.
- To get logs and store them in the child appdomain.
- To use getLog() function to retrieve test logs from the appdomain.
- To retrieve test results and test logs for every test execution along with the code developer’ s identity and current time for display.

2. Organizing Principles

The organizing principles are to perform test requests in queue order, decode the XML files for address of DLL files, execute the test request in isolate child appdomain, use test () to return and store the results, logs etc. and use getLog() function to retrieve the logs.

3. Key Structural Ideas

- To build a test queue which enqueues the received Test Requests, and perform the tests serially in dequeued order.

- the Test Harness creates independent child appdomain for each test request.
- To decode the XML files and load the DLL files. Every test driver and the test library are implemented as a DLL file.
- Each child appdomain contains a test loader. Test loader can load the test driver of the DLL files, and use the method test () declared by ITest to execute test on the code. Test () returns the pass status and test results, and it stores the results in repository.
- ITest package contains the getLog() function which returns a string presentation of test log. GetLog method can retrieve the test logs stored by appdomain.
- The isolate appdomain protects the Test Harness processing from being influenced by the crash of the individual test processing.

3. Uses

1. Myself

I am one of the primary users, when I finish it, I will run test code on it to make sure it meets the requirements. I can perform tests on Test Harness, making test requests, each in the form of an XML file which contains test driver and test code, then examine the outputs.

I will also use this version of Test Harness for the developing of project 4 in this semester. In project 4 I will probably expand it into a Test harness client and implement a Test Harness Sever that accepts Test Requests from the clients.

2. Instructor and TAs

This project will be submitted to instructor and teaching assistants for grades. They will read the documents of it, run test requests on it and examine whether all the requirements have been successfully accomplished.

3. Developers

Developers especially those who develop large systems, will use Test Harness to do integration tests on the packages, modules etc. created by themselves. They can get helpful debug information from the test logs and results which can help developers make sure that the individual’s works can fit into the baseline of large project.

The developers need the test logs for debug information, their interface will demonstrate more detailed test logs.

4. Quality Assurance Engineers

The quality assurance engineers will use Test Harness for integration test to guarantee that every part of the project performs well in the baseline.

The quality assurance engineers concentrate on the test results and the success rate. Their interface will demonstrate the results of test more than the logs.

5. Project managers

Project managers will use Test Harness to capture general information of their projects. They can read the test logs and results and understand the current status of the projects conveniently.

The project managers will probably test big systems on Test Harness, so the system should consider performance and storage of big data as an issue.

4. Application Activities

Activity diagrams are drawn to show the detailed steps of the application execution thus apparently illustrate the function of the application.

The Test Harness processing shall be divided into 7 activities. The high level activity diagram is expressed in below diagram. The detailed activity diagram will be showed in another diagram.

4.1 Read and enqueue the XML files

First activity of the system shall be getting input from users. Test executive is the entry of Test Harness. Test Executive reads the XML files into string format for saving space in queue. Then Test Executive calls the Queue package to enqueue the XML files. The XML files shall not dequeue until Queue receives request from child appdomain.

4.2 Create the child appdomain

The execution of DLL files test will run in child appdomain, isolated from the processing of Test Harness. So before XML file dequeues, appdomain manager shall create a new child appdomain, and send request to Queue for new XML file. When Queue receives the request, it will dequeue the XML file which is in string format, then send it to the child appdomain.

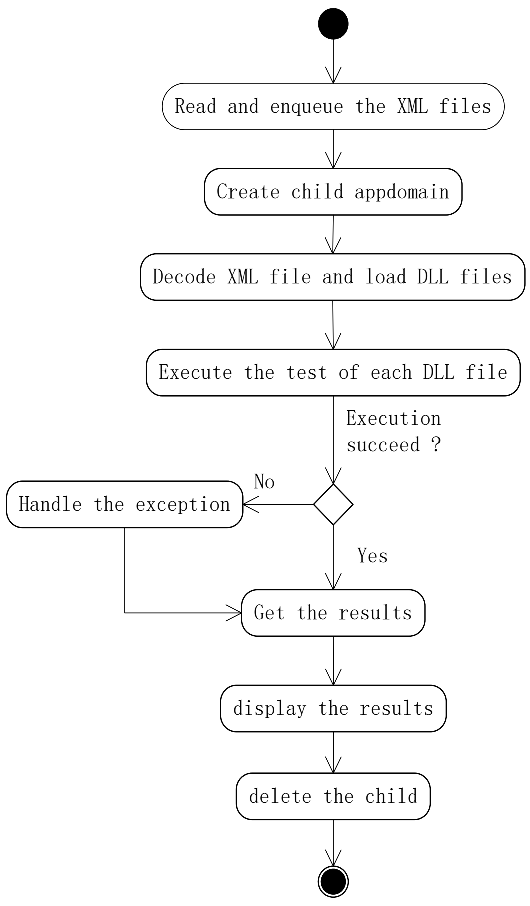


Figure 1High level activity diagram of Test Harness

4.3 Decode the XML files and load the DLL files

XML string shall be decoded by the XML Decoder firstly. XML Decoder will decode the XML string to application readable format. Test Executive will call the File Manager to search the directory tree in XML file and display the names of all DLL files, then fetch the DLL files from repository for further processing.

4.4 Execute test processing of each DLL file

When DLL files are from When DLL files are fetched from repository, Test Executive calls the Test Loader to load the DLL files. Test Loader will read the DLL file’ s test driver, and load the test library which contains the test code. There shall be a loop in the child appdomain to fetch DLL files in the order determined by the XML file, and test loader load the test library in multiple threads for test processing. The test() method in ITest can be call by Test Executive, taking no arguments and returning the test pass status, e.g., a Boolean true or false value. The getLog() function in ITest can be used to retrieve logs from the child appdomain.

4.5 Handle the exception

Each child appdomain shall be isolated with Test harness appdomain. The only way they communicate is call the IHello and IReply interface. This guarantees the exceptions in child appdomain when run the test will be well handled without bother the Test Harness processing.

Exception Monitor uses the **try**, **catch**, and **finally** keywords to try actions that may not succeed, to handle failures when you decide that it is reasonable to do so, and to clean up resources afterward. Exceptions can be generated by the common language runtime (CLR), by the .NET Framework or any third-party libraries, or by application code. Exceptions are created by using the throw keyword. [1]The results will be sent to Display package to be displayed to the console.

4.6 Get the results

Results and logs are stored by the child appdomain. Test Executive can use getLog() function in ITest interface to retrieve logs from child appdomain. Results and logs shall be merged as test results to be displayed according to users' mode.

[1]" C# Programming Guide". https://msdn.microsoft.com/en-us/library/ms173160.aspx (Sep. 13,2016)

4.7 Display and end the test execution

Test results shall be sent to Display package to be displayed to the console. When results are successfully displayed, child appdomain will use Queue Query package to query whether the queue is empty. If the queue is not empty, child appdomain shall use IHello interface to send a message to request appdomain of Test Harness to delete the child appdomain and create a new one for the next XML file in the queue. If the queue is empty, child appdomain shall use IHello interface to send a message to request appdomain of Test Harness to end the process.

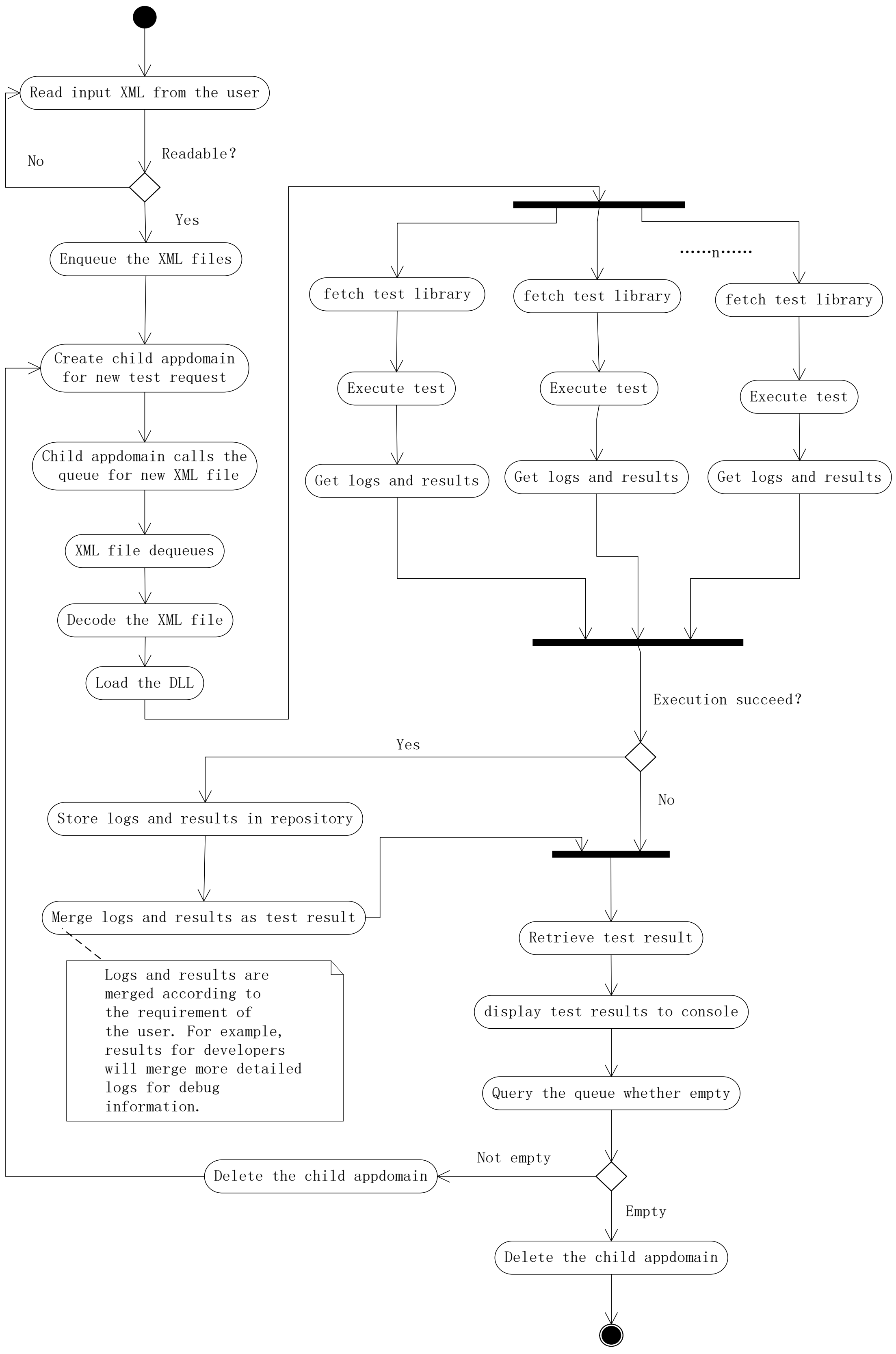

Read input XML from the user
Readable?
No
Yes
Enqueue the XML files
Create child appdomain for new test request
Child appdomain calls the queue for new XML file
XML file dequeues
Decode the XML file
Load the DLL
fetch test library
Execute test
Get logs and results
……n……
fetch test library
Execute test
Get logs and results
fetch test library
Execute test
Get logs and results
Execution succeed?
Yes
No
Store logs and results in repository
Merge logs and results as test result
Logs and results are merged according to the requirement of the user. For example, results for developers will merge more detailed logs for debug information.
Retrieve test result
display test results to console
Query the queue whether empty
Not empty
Delete the child appdomain
Empty
Delete the child appdomain

Figure 2 Detailed activity diagram for Test Harness

5. Partition

Test Harness is divided into several packages, each of them conduct its own responsibility. The performance of Test Harness depends on the interactions among these packages.

Figure 3 is Package Diagram of Test Harness.

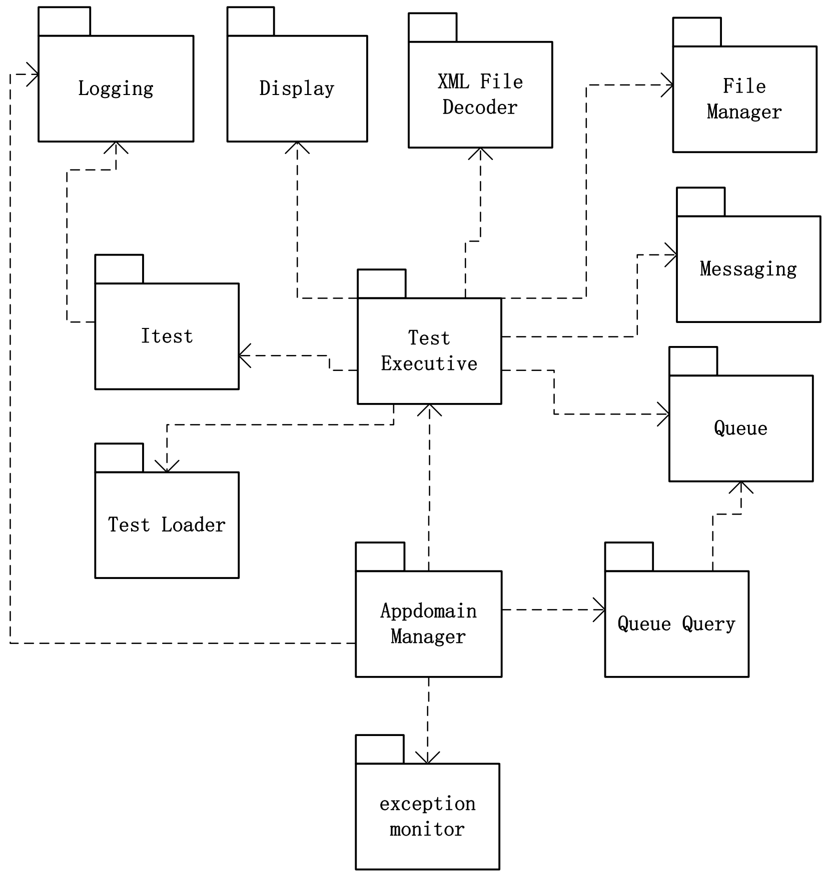


Figure 3 Package Diagram of Test Harness

5.1 Test Executive

Test Executive is the primary package of the system, which controls various interactions with other packages. The Test Executive holds the main method, and is responsible for the main control flow of the system, beginning with reading input as XML files and ending in calling display package to demonstrate the results. The packages Test

Executive calls while conducting tests are as follows:

- XML decoder package to decode the XML files and build queue for loading.
- Appdomain Manager to create the child appdomain for the test execution, and delete it when test finishes.
- Logging package to get test logs and store logs by child appdomain.
- ITest package to call the test() function, executive the test and return the results.
- Display package to demonstrate the results to the command line.
- Messaging package to pass the results and logs to other packages.

5.2 File Manager

File manager can be used to search the directory tree in XML files and display the names of all DLL files. File Manager will be called by the Test Executive to find the DLL files in the repository, each with one test driver and one test library. DLL files will be further processed in the child appdomain.

5.3 XML Decoder

When new XML file dequeues and be sent to the child appdomain, this package will be called by the Test Executive to decode the input XML files. It decodes the XML files into application readable format, passes the results to Test Executive for further processing in child appdomain.

5.4 Queue

Queue package is used to build a queue for the XML files. Queue builds the queue before Test Executive read the input XML files, then enqueues the XML files. Queue will dequeue new XML file for test exception when receives the dequeue require from child appdomain.

5.5 Appdomain Manager

Appdomain Manager is used to create child appdomain for a test request. Child appdomain requires the Queue to dequeue new XML file, and isolates the test processing from the Test Harness processing.

Appdomain Manager will query the Queue whether queue is empty. If the queue is not empty, Appdomain Manager will call Test Executive to delete the child appdomain and create a new one for next test request. If the queue is empty, Appdomain Manager will call Test Executive to delete the child appdomain and end the test execution.

5.5 Test Loader

When DLL files are retrieve from the repository, Test Executive calls the test loader to load the DLL files for further processing.

5.7 Logging

Logs are very important test information in this system. Logging package declares methods to get logs of the test execution and store the logs by the child appdomain. Test logs can be retrieved by getLog() function in ITest.

5.8 ITest

ITest is the package which contains the main functions to execute test on test code. The test() method in ITest can be call by Test Executive, taking no arguments and returning the test pass status, e.g., a Boolean true or false value. The getLog() function in ITest can be used to retrieve logs from the child appdomain.

5.9 Messaging

Messaging package is used to pass the commands, results and logs to other packages. For example, when the child appdomain queries the XML files queue, the queue package uses the messaging package to send the result of the query.

5.10 Display

Display package is used to receive message which contains the results of test from Test Executive, and display the results to console. The DisplayRes() function in the package defines the format of console output.

5.11 Exception Monitor

The test execution may crash because of exceptions. Each test request process is isolated in child appdomain to protect Test Harness from being influenced by it.

Exception Monitor package is used to catch exceptions in child appdomain, document the exception information and store it in the repository as test results. Exception monitor will stop the test execution and delete the child appdomain when it catches the exception.

5.12 Queue Query

Queue Query package will be called by child appdomain to query the queue whether it is empty. This package will query the Queue package, and use Messaging package to pass the results to child appdomain.

5 Critical Issues

1. Test Requests handling

What kinds of test request can be accepted by the Test Executive? How to deal with the test request? Test Executive should take the input test request, and provide the order of test execution, decode the request into application readable format.

This issue concerns the entry of the system, and is of vital importance.

Solution:

- The Test Executive only accept test request in the form of a XML file. Assuring that the test request can be read by Test Harness.
- XML Decoder package will decode the XML files into application readable format for further processing.
- File Manager will search the directory tree in decoded XML files and find the DLL files in the repository, each with one test driver and one test library. DLL files will be further processed in the child appdomain.

Test driver example:

```
<?xml version="1.0" encoding="utf-8" ?>
<testRequest>
  <author>Jim Fawcett</author>
  <test name="First Test">
    <testDriver>td1.dll</testDriver>
    <library>tc1.dll</library>
    <library>tc2.dll</library>
  </test>
  <test name="Second Test">
```

```
        <testDriver>td2.dll</testDriver>
        <library>tc2.dll</library>
      </test>
    </testRequest>
```
[2]

2. Performance

Every system of Large scale should take performance into consideration. Performance issue in this system concerns about how many Test Requests it can deal with, how many DLL files a Test Request can contain, how long it takes to conduct test execution on a test request with several DLL.

When the loads are typical, estimated just a few XMLs in the queue and each of them just contains a few names of the DLL files, performance of the system will be not challenged. But when loads are extreme, for example at the deadline of a task, more than a hundred Test Requests will squeeze into the queue. What' s more, system will probably test really big project which contains hundreds of DLL files. Performance will become an issue of much importance.

Solution:

- The package Queue is designed to conduct enqueue and dequeue operations with input XML files. In case the amount of XML files is really large, Queue will allocate large space to create the queue. And the XML file will not be dequeued until Queue receives requirement from child appdomain.
- Test Executive will allocate enough space for Child appdomain to deal with a Test

[2] Jim Fawcett "XML help code" . http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/code/Project2HelpF16/XMLHelp/(Sep.13.2016)

Request with hundreds of DLL files. Test Loader package is designed to load the DLL files.

- When XML files are in queue, they will be stored in string format. Because the string format need less space than the XML files. DLL files in the repository will not be retrieved until the
- Test Executive will delete the child appdomain when a test request finishes the test execution. This operation will free the space for new appdomain., and prevent Test Harness from being crashed by the accumulating load.

3. Exception handling

Test Harness must be robustness and exception-tolerant system. How to handle the exceptions and prevent Test Harness processing from being influenced?
Every successful system should consider solutions to this issue.

Solution:

- Test request’s execution will be isolated into an appdomain that Test Harness processing will not be bothered by the exceptions in execution of test request.
- In the appdomain, when any unexpected or exceptional situation occur, Exception Monitor package will use The C# language's exception handling features to deal with exceptions.

Exception Monitor uses the **try**, **catch**, and **finally** keywords to try actions that may not succeed, to handle failures when you decide that it is reasonable to do so, and to clean up resources afterward. Exceptions can be generated by the common language runtime (CLR), by the .NET Framework or any third-party libraries, or by application code.

Exceptions are created by using the throw keyword. [3]The results will be sent to Display package to be displayed to the console.

- Executions in the child appdomain will be well managed, but executions in the baseline of Test Harness will straight crash the system. So code in the baseline, for example: load test code in the Test Loader package, catch and finally code in Exception Monitor package should be really short and clear, avoid dangerous code like pointers, guaranteeing the success rate.

4. Demonstration

How will the system demonstration the results and logs to be conveniently and clearly read by the users? What results should be demonstrated to the console? Test Harness should demonstrate different results to meet different users' requirement.

Solution:

Test Harness will retrieve the results and logs and merge them before demonstration in the command line. Different users' mode will merge the results in different ways. For example, developers will ignore the success results and concentrate on the failure and the logs. Quality assurance will need the success rate of the DLL files.

5.Unreadable Input

If unreadable input is accepted, errors will occur when input is read by the application. Test Harness must handle the unreadable input.

Solution:

[3]" C# Programming Guide". https://msdn.microsoft.com/en-us/library/ms173160.aspx (Sep. 13,2016)

Test Harness should declare that the input test requests should be in the form of XML files. The get input function will not accept test requests in other format except XML files.

6.User Interface

User Interface should be clear and convenient. And different user should choose different mode to execute the test.

Solution:

This system is only the first version Test Harness, so user interface will be the command line for convenience. User can choose the user role for three modes: student/instructor, developer, quality assurance/project manager. Command line shall be given as below

/u for choosing user mode

/p for input XML files

/r for displaying the results

/q for ending the test execution

6. Conclusion

In conclusion, Test Harness can receive test requests, do the test successfully and demonstrate results clearly.

The OCD illustrates the system aimed users and what they can do with Test Harness. Different modes that meet different requirements of users are discussed. Test Harness is divided into 12 packages to perform the function.

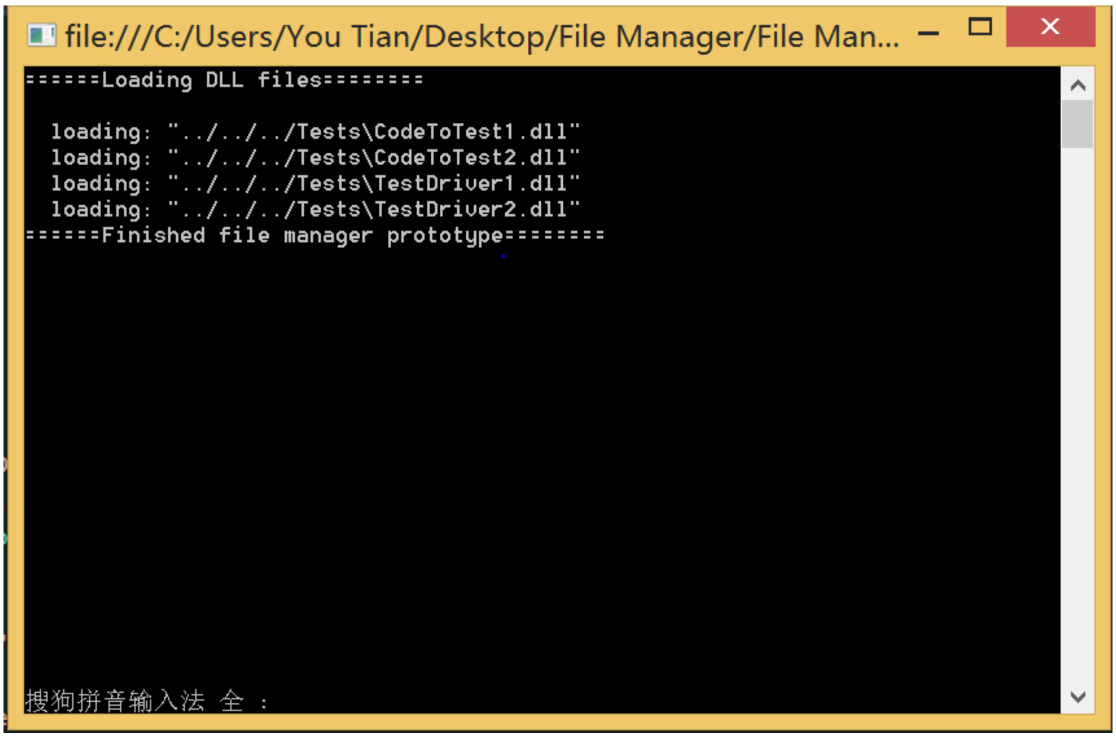
The OCD also present detailed activities in the Test Harness processing. OCD illustrates how the test execute in the system and help us understand the system.

7. Appendixes

File manager prototype:

File manager prototype is File Manager package that searches a directory tree, rooted at a specified path, and displays the names of all DLL files encountered. The source code is attached to the document as files.

The output of file manager prototype is as below:



```
======Loading DLL files=========

  loading: "../../../Tests\CodeToTest1.dll"
  loading: "../../../Tests\CodeToTest2.dll"
  loading: "../../../Tests\TestDriver1.dll"
  loading: "../../../Tests\TestDriver2.dll"
======Finished file manager prototype========
```

Figure 4 File Manager output

Child appdomain loading test library prototype:

Child appdomain loading test library prototype runstests by loading DLLs and invoking test(). It uses the try and catch keywords to try test load process, to handle failures. The source code is attached to the document as files.

8. Reference

http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/code/Project2HelpF16/XMLHelp/

https://msdn.microsoft.com/en-us/library/ms173160.aspx

http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/lectures/Project1-F2016.htm#top