Build Server

Operational Concept Document

CSE681-PROJECT #4

By Chai Weiheng (SUID: 207547980)

Instructor: Jim Fawcett

Date: 28th November, 2017

Contexts

[1. Executive Summary 4](#_Toc500357951)

[2. Introduction 5](#_Toc500357952)

[1. Application Obligations 5](#_Toc500357953)

[2. Organizing Principles 6](#_Toc500357954)

[3. Key Architectural Ideas 6](#_Toc500357955)

[3. Users and Uses 8](#_Toc500357956)

[1.Users and Uses 8](#_Toc500357957)

[1.1Myself 8](#_Toc500357958)

[1.2Instructor and TAs 8](#_Toc500357959)

[1.3programmer 8](#_Toc500357960)

[1.4QA 8](#_Toc500357961)

[2.User interface 8](#_Toc500357962)

[4. Structure 10](#_Toc500357963)

[4.1Introduction of the whole system 10](#_Toc500357964)

[4.1.1architecture 10](#_Toc500357965)

[4.1.2activity 12](#_Toc500357966)

[4.2Commservice 14](#_Toc500357967)

[4.3Mock Client 15](#_Toc500357968)

[4.4Repository 16](#_Toc500357969)

[4.4.1structure 16](#_Toc500357970)

[4.4.2activities 16](#_Toc500357971)

[4.5Build Server 18](#_Toc500357972)

[4.5.1Mother Builder 20](#_Toc500357973)

[4.5.2Child Builder 22](#_Toc500357974)

[4.6Testharness 25](#_Toc500357975)

[4.6.1 structure 25](#_Toc500357976)

[4.6.2activities 26](#_Toc500357977)

[6.Critical Issues 27](#_Toc500357978)

[1. I never write any code before I come here, how can I implement the project before the deadline 27](#_Toc500357979)

[2. Running Steadily 27](#_Toc500357980)

[3. Demonstration 27](#_Toc500357981)

[5.Thread safe 28](#_Toc500357982)

[6.The Repository sometimes will crash if I select several Build Requests to build at the same time 28](#_Toc500357983)

[7.Performance 28](#_Toc500357984)

[8.When the Child Builder receive the Build Request, the Child Builder will parse it and send file request to the Repository, then build the files. But when the Child Builder start to build, some needed files maybe not received. 28](#_Toc500357985)

[7.Conclusion 29](#_Toc500357986)

[1. Appendixes 30](#_Toc500357987)

[1.1The structure of the CommMessage 30](#_Toc500357988)

[1.2Changes and Deficiencies 31](#_Toc500357989)

[2. Reference 31](#_Toc500357990)

# Executive Summary

This development will create a Build Server, capable of building C# libraries, using process pool to conduct multiple builds in parallel. The implementation is accomplished in three stages.

The first, Project #2, implements a local Build Server that communicates with a mock Repository, mock Client, and mock Testharness, all residing in the same process. Its purpose is to allow the developer to decide how to implement the core Builder functionality, without the distractions of communication channel process pool.

The second, Project #3, develops prototypes for a message-passing communication channel, a process pool, that uses the channel to communicate between child and parent Builder, and a WPF client that supports creation of build request messages.

Finally, the third stage, Project #4, completes the build server which communicates with a mock Repository, mock Client, and mock Testharness, to thoroughly demonstrate Build Server Operation.

The final product consists of a relatively small number of packages. For most packages there already exists prototype code that show how the parts can be built. For this reason, there is very little risk associated with the Build Server development.

Critical issues include: implementing the project before the deadline, the performance of the system, the safety of the system and many other critical issues. All of these issues will be discussed.

The Build Server will function as one of the principle components of a software Development Environment Federation, the others being Repository, Testharness, and Federation Client. Building these other Federation parts is beyond the scope of this development.

# Introduction

Todays the developers may have to develop very large systems. In order to successfully implement big systems we need to partition code into relatively small parts and thoroughly test each of the parts before inserting them into the software baseline2. As new parts are added to the baseline and as we make changes to fix latent errors or performance problems we will re-run test sequences for those parts and, perhaps, for the entire baseline.

Before we test the small parts and add them to the baseline, we have to build them into libraries, the Build Server will help with this.

## Application Obligations

The primary requirement of Build Server is building the code into library. So that the code can be tested by the Testharness and the libraries can be stored in the Repository. The main obligations of the application should be:

* To accept one or more Build Requests, each in the form of an XML file.
* Using process pool to create several child processes to make the build work efficient.
* To parse the Build Requests.
* To get files from the Repository according to the information in the Build Request.
* To build the files from the Repository into libraries according to the Build Requests, send the libraries to the Repository to store and send the libraries to the Testharhess to test.
* Creating build logs and send it to the repository.

For other part of the system, the Client should have a GUI so that the user can control the system’s work with the GUI, the users can browse the files in the Repository, create Build Request and ask the Repository to send Build Requests to the Build Server by the GUI. The Repository should store the libraries, Build Requests, logs, and send Build Request to the Build Server. The Testharness should test the library and send logs to the Repository.

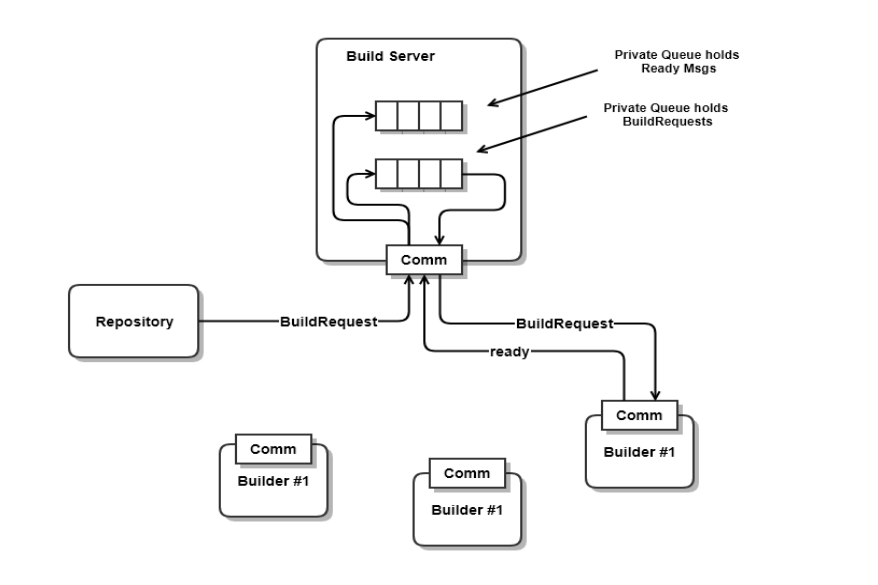
## Organizing Principles

The main idea of my design is to use message to drive the action of each server of the system, for example, the build server will start the build work when it receive a message tell it which Build Request needed to build, communication is the most important thing for the whole system, using WCF to communicate between every server in the repository and between the servers and the Client.

The design of the Build Server is to use the MSBuild, build the files into library, and using the FileLog class, create logs in txt file.

## Key Architectural Ideas

The build server may have very heavy work loads just before customer demos and releases. We want to make the throughput for building code as high as is reasonably possible. To do that the build server will use a "Process Pool". That is, a limited set of processes spawned at startup. The build server provides a queue of build requests, and each pooled process retrieves a request, processes it, sends the build log and, if successful, libraries to the test harness, then retrieves another request. I use figure 1 to show the idea clearly, and the author of figure 1 is Dr. Fawcett.



*Figure 1 Builder Server*

In my design, the Mother Builder is the build server in the figure 1, the Child Builder is the pool process in the process pool. The Build Server should have two major parts, the first is the Mother Builder, and the second is the child builder(process pool), the process pool can start a number of child builder according to the user’s command, and send the Build Request to the child builder. Each child builder can parse the Build Requests, get files from the Repository, and build the library. The Mother Builder has two private queues, one stored the ready message from the Child Builder, another stored the Build command from the Repository, or the close process pool command by the Client. When the two queue are both empty, deque both two queues, if the command is build, send the Build Request to the Child Builder which is ready, else if the command is close, close the ready Child Builder

The whole system also includes Client, Repository, Build Server and Testharness.

# Users and Uses

## 1.Users and Uses

### 1.1Myself

I am one of the users, when I finish it, I test the whole system several times to make sure that it meets all the requirement.

Impacts on design: to meet the requirement, I check every details of the code to make sure that I meet the requirement.

### 1.2Instructor and TAs

This project will be submitted to instructor and teaching assistants for grades. They will read the documents of it, test the whole system to see if I meet all the requirement.

Impacts on design: I show how I meet the requirements clearly in the console.

### 1.3programmer

The programmer will use the Build Server to build the code need to be tested into library. After the code were built into the libraries it will be stored in the Repository, so it can be reused.

Impacts on design: we should think about how to build different language files. And we should make the system as efficiency as possible.

### 1.4QA

The QA will use the Build Server to see if the program can be built into library successfully, and want to see the details of the build logs.

Impacts on design :I try to design the code to create the build logs to show the details .

## 2.User interface

In the project #4, I used the wpf to design a GUI(Graphical User Interface), the GUI make the whole system ease to use, the user can see the files in the Repository, and can selected the files that should be built into a library By the GUI, I will talk about the GUI in the structure part.

# Structure

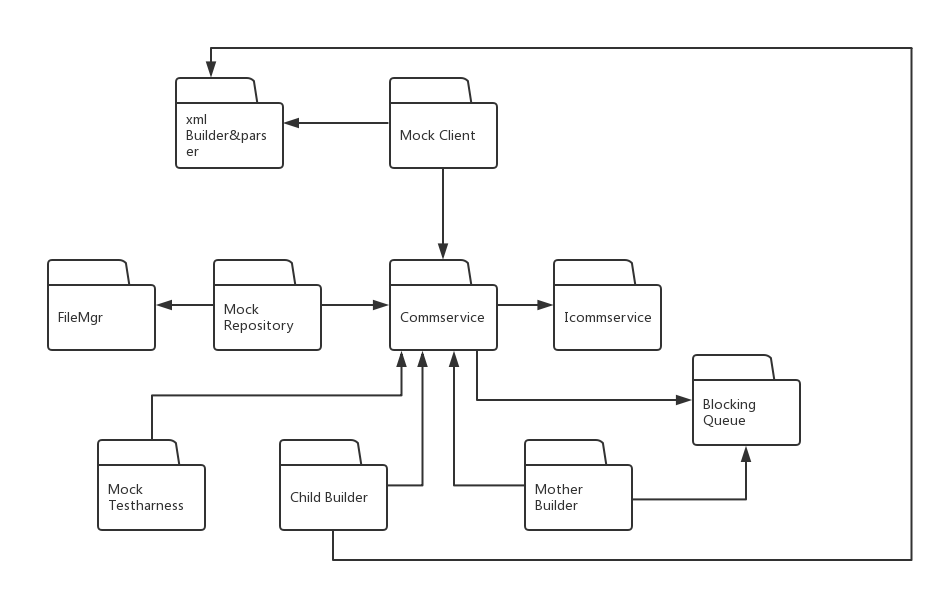
**4.1Introduction of the whole system**

Every action of the system is driven by the message, for example, the build server will not work even the repository send a Build Request file(the xml file) to it until it receive a message that gives it a command to build. The structure of the message will be showed in the appendix.

**4.1.1architecture**

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

I show the structure of the whole system by the Figure 1.



*Figure 2 Package Diagram of the whole system*

1.Mother Builder

The Mother Builder is an important part of the Build Server, it can start a number of Child Builder, and close the Child Builder according to the message from the Client. The Mother Builder can send Build Request to the Child Builder which is unoccupied. The Mother Builder has two private queues, when the Child Builder send a ready message to the Mother Builder, the Mother Builder will enqueue the message in one of the queue, and when the Repository or the Client send a message to the Mother Builder, the Mother Builder will enqueue the message into another queue. When the two queues are both not empty, the Mother Builder will dequeue the two queues to know which Child Builder is ready, and send the Message to this Child Builder. The Mother Builder uses the commservice to do send and get messages and files.

2.Child Builder

Child Builder is another important part of the Build Server, it can get Build Request from the Mother Builder, and parse the Build Request. After parse the Build Request, the Child Builder will send a request to the Repository to get the tested files. After the Tested files are sent to the Child Builder, the Child Builder will try to build it, create build log, and send build logs to the Repository. If build success, the Child Builder will send the library to the Repository and the Testharness, and send a message to the Testharness to tell it which library to test, then send a ready message to the Mother Builder. If build failed, just send a ready message to the Mother Builder. The Child Builder calls the xml builder to parse the Build Request, and uses the commservice to get and send messages and files.

4.Mock Client

The Mock Client include the GUI, it is an important part of the whole system. The users use the GUI communicate with the system. The Mock Client uses WPF to design the GUI. Client uses message dispatcher to get the list of the files in the Mock Repository and shows it in the GUI, and uses the xml Builder to create the Build Request. The user can use the send button ask the repository to send the selected Build Request to the Mother Builder. The Mock Mock Client uses the commservice to get and send messages and files.

4.Mock Repository

The Mock Repository is the main storage of the whole system. The repository can store the Build Requests, the build logs, the test logs and the libraries. It can send the Build Request in its storage to the Mother Builder according to the message from the Client, and send the tested files to the Child Builder according to the message from the Child Builder. The Mock Repository uses the FileMgr package to get the list of the files and the dirs, and uses the commservice to get and send messages and files.

5.Mock Testharness

The main task of the Mock Testharness is to test the libraries sent by the Child Builder. After the library being tested, the Testharness will send the test log to the Repository. The Mock Testharness uses the commservice to get and send messages and files.

6. Commservice

Commservice provides communication services used by MotherBuilder and ChildBuilders.

7.ICommservice

ICommservice defines the CommMessage structure used by communicators.

8.FileMgr

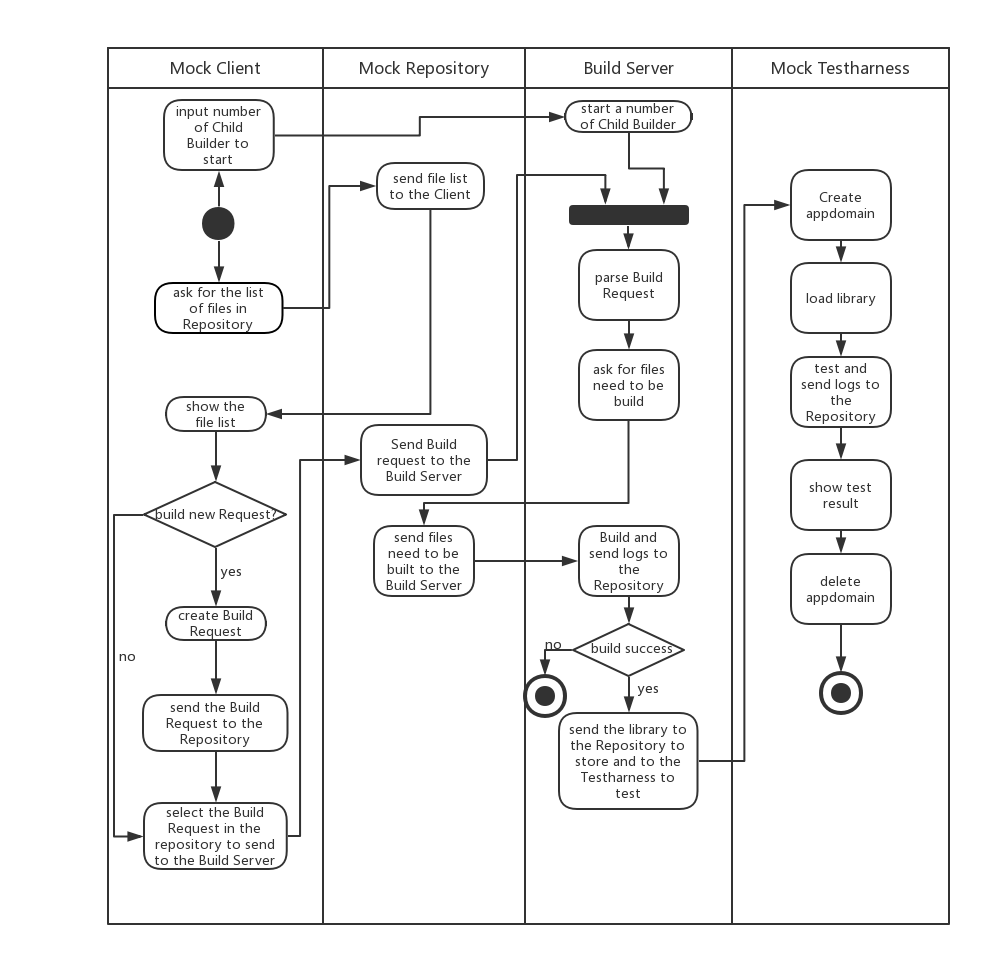
FileMgr provides methods to get the list of the files.

9.xml Builder

The xml Builder provide methods to create Build Request and parse Build Request.

### 4.1.2activity

Here I show the activity of the whole system by the Figure 2.

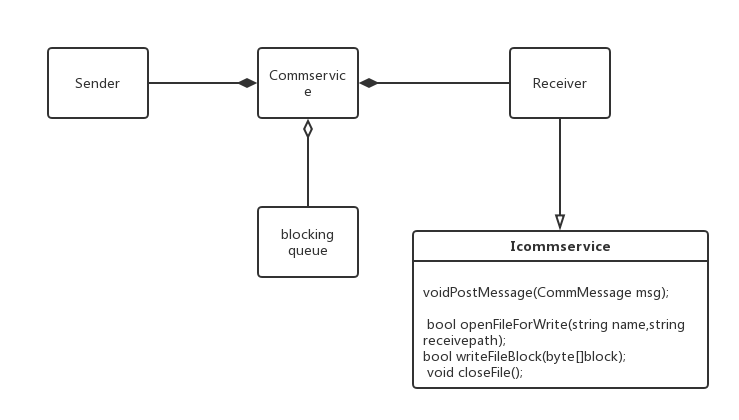


*Figure 3 Activity Diagram of the whole system*

To show how the system’s work clearly, the activity diagram only shows how the system build and test with one Build Request. The Client can control the Build Server start a number of Child Builder. When start the work, the Client will send a message to the Repository to get the list of files in the Repository, the Repository will send the list of files and dirs to the Client, then the Client will show them in the GUI. When the user sees the list of files, the user can decide to create a new Build Request or use the Build Requests in the Repository. And then ask the Repository to send the Build Request(contains information about the files the user want to build) to the Build Server. After the Child Builder(s) are started, and the Build Server receive the Build Request, the Build Server will parse the Build Request, and then get files from the Repository. After the Repository send the needed files to the Build Server, the Build Server will try to Build them and send logs to the repository. If build failed, the work is end. Else, send the library to the Repository to store and to the Testharness to test. When the Testharness receive the library to test, it will create a appdomain and then load and test the library. The Testharness will also send test logs to the Repository to store.

## 4.2Commservice

The servers in the system using the Commservice to send(get) files and messages from each other. This class diagram shows the structure of the Commservice.

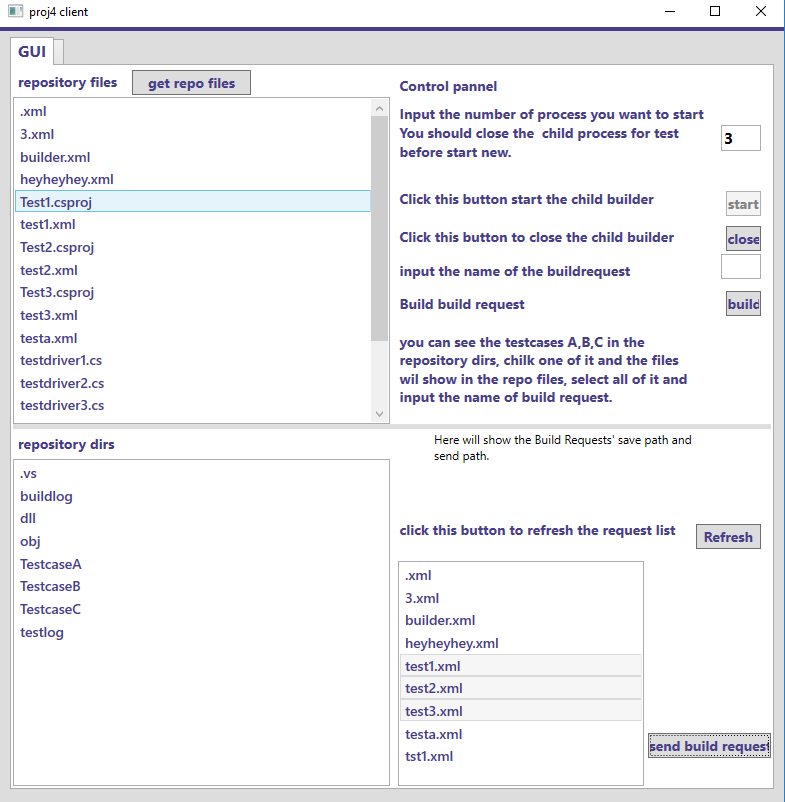


*Figure 3 Class Diagram of the Commservice*

All the servers create instance of sender and receiver to communicate, the sender is used for sending messages and files, and the receiver can create host with different port to get messages and files. The using of blocking queue is really important, it makes the communicate Thread safe. The ICommservice defines the CommMessage structure used by communicators. The messages passed by different are all the instance of CommMessage!

## 4.3Mock Client

The Mock Client include the user interface, the GUI. Users use the GUI to control the whole system’s work, the figure shows what the GUI looks like:



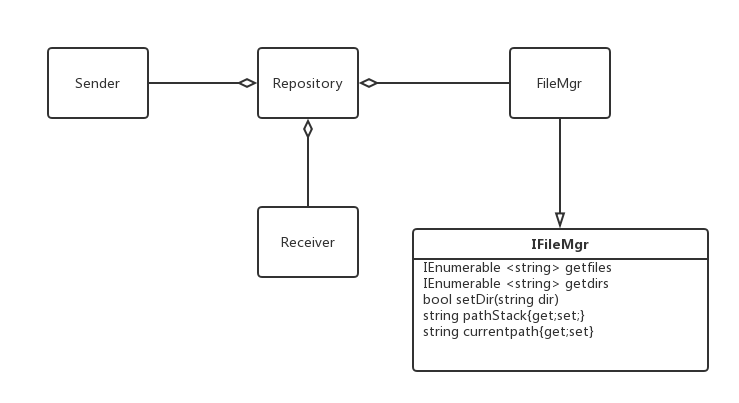
*Figure 4 The GUI*

In the Client, I use message dispatcher to get the list of files in the repository, and show them in the list box, the first list box shows all the files in the Repository, and the second shows the dirs in the Repository, the user can select the files in the list box and click the build button, the new Build Requests will be showed in the third list box, which only shows the xml files. When you select the files in the xml list box and click send build request, the GUI will send a message to the Repository, and the Repository will send the Build Request to the Mother Builder. When the user clicks the close button, the GUI will send a message to the Mother Builder, and the Mother Builder will close all the Child Builders, then the start button can be used again, the user can click it to start new Child Builder process.

## 4.4Repository

### 4.4.1structure

The Repository is the storage of the whole system. Here is the class diagram of the Mock Repository to show the structure of the Repository.

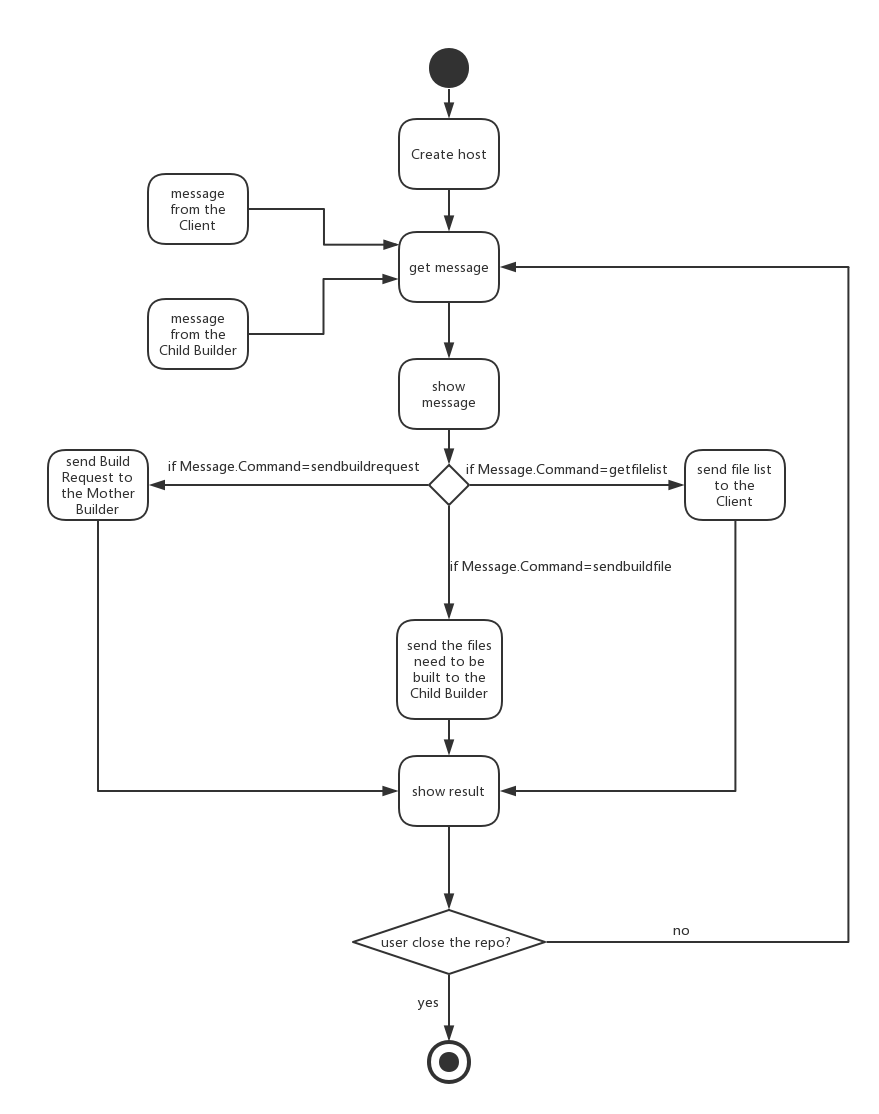


*Figure 5 Class Diagram of the Repository*

The Repository has a simple structure, the diagram shows the relationships between the classes in the Repository, the Repository has a Receiver and two Senders, one connect to the Mother Builder and another connect to the Client.

### 4.4.2activities

The activity diagram shows how the system works:



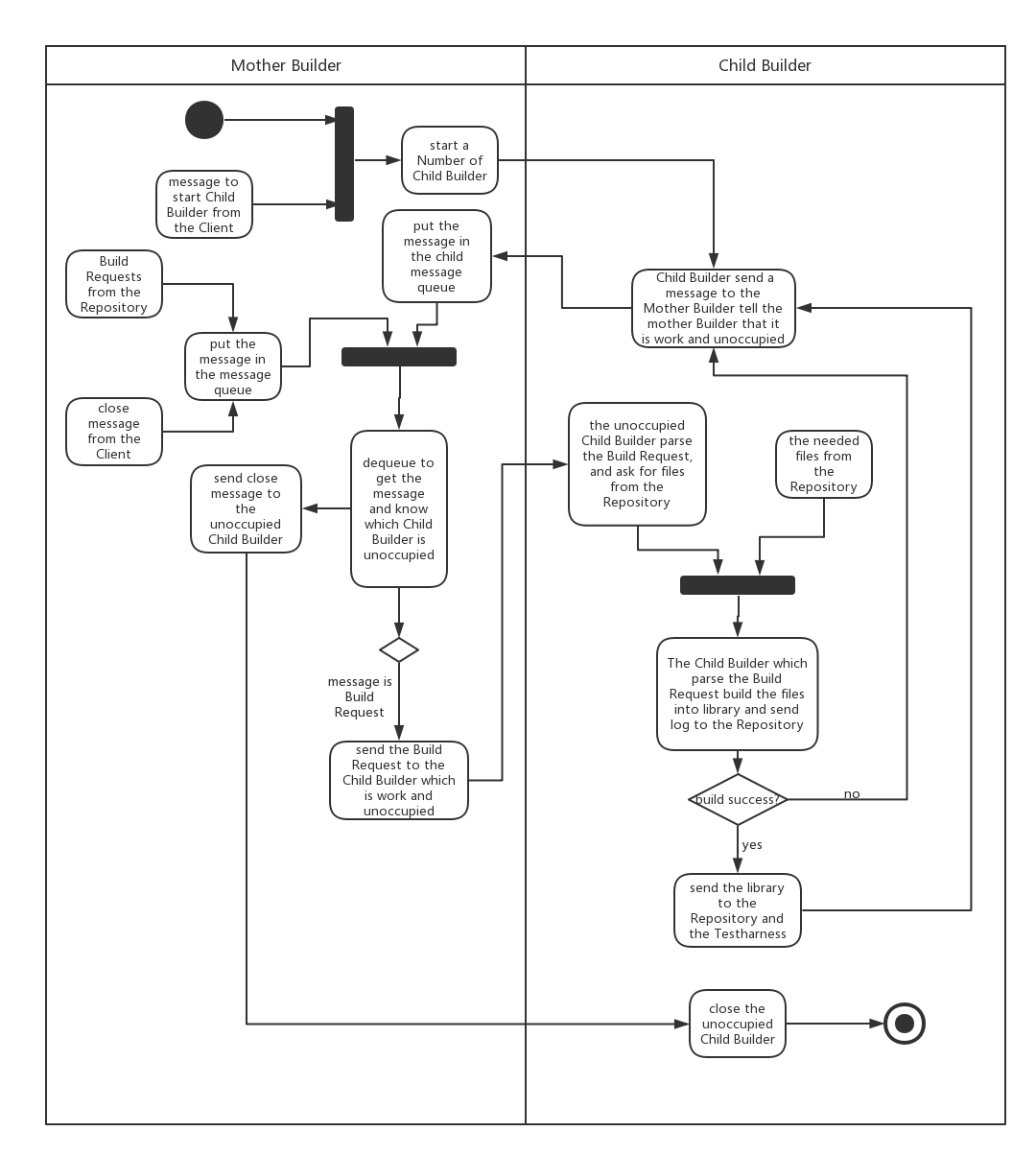
*Figure 6 Activity Diagram of the Repository*

The Repository create a host to receive message and files, It will receive three different kinds of CommMessage, the first is the Client’s request to get the list of files in the Repository. The second is the Client’s request, ask the Repository to send Build Request to the Mother Builder, when the Repository get this message, the Repository will send the Build Request file to the Mother Builder, and then send a CommMessage to the Mother Builder, the Commmessage’s body is the name of the Build Request to let the Mother Builder know which Build Request to execute.

## 4.5Build Server

The main task of the Build Server is to build the files into libraries.

The Build Server include the Mother Builder and the Child Builder(the pool process), the Mother Builder will start the Child Builders, get Build Requests, and send Build Requests to the ready Child Builder. The Child Builder will parse the Build Request and get files from the Repository, try to build the files into libraries, show the result in the console, send logs to the Repository, and send libraries to the Testharness and the Repository. I will show how the whole Build Server work, and then discuss them separately. The diagram below shows the activity of the Build Server.



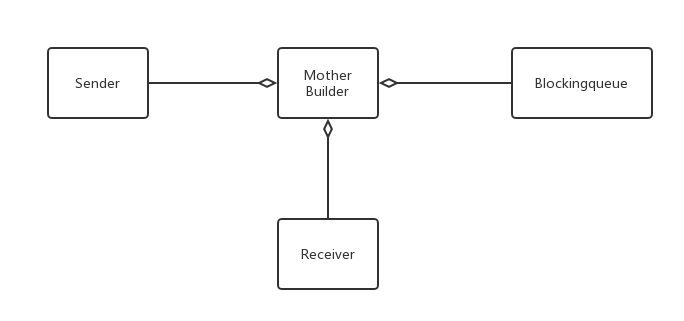
*Figure 7 Activity Diagram of the Build Server*

The Mother Builder get message from the Client, and start several Child Builder According to the message. And then every Child Builder send a message to tell the Mother Builder that it is not occupied, the Mother Builder will also get the messages from the Repository and Client. If the two queues are both not empty, the mother Builder will dequeue the two queues to know which Child Builder is not occupied and to know the message. If the command of the message from the Repository and the Client queue is close, the Mother Builder will send a close command to the Child Builder which is the source of the ready message. Else, the Mother Builder will send the Build Request(both the xml file and the name of the file) to the Child Builder and the Child Builder will parse the Build Request, then get the files from the Repository, try build the files into library and send log to the Repository. If build success, send library to the Repository and Testharness, and send message to the Mother Builder to tell the Mother Builder that it is unoccupied. Else if build is not success just send a ready message to the Mother Builder.

### 4.5.1Mother Builder

4.5.1(a)structure

This class diagram shows the structure of the Mother Builder:

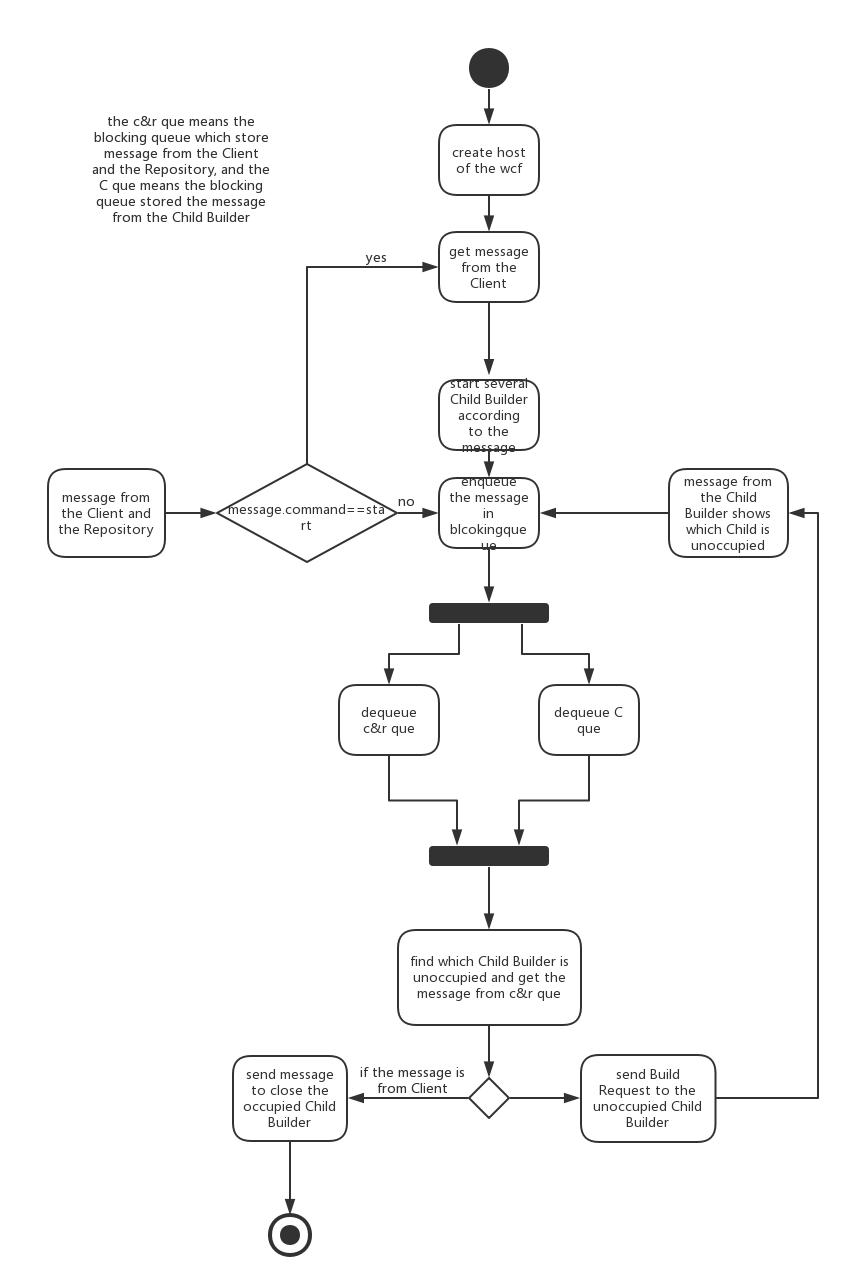


*Figure 8 Class Diagram of the Mother Builder*

The mother Builder has a Receiver, a Sender and two blocking queue. The Receiver will get the messages from the Client, the Repository and Child Builder. The Sender will send messages and files to the Child Builder. One blocking queue will process the messages from the Client and Repository, we call it C&R que, another will process the messages from the Child Builder, we call it C que here.

4.5.1(b)activities

The activity diagram below shows how the mother builder work:



*Figure 9 Activity Diagram of the Mother Builder*

The Mother Builder create a host and then get messages. The messages’ command from the Client can be start Child Builder or close the Child Builder, if the command is start, the Mother Builder will use process().start to start the Child Builder according to the message, without enqueue the message into the C&R que. Else if the command is close, the Mother Builder will enqueue the message to the C&R que. The command of the messages from the Repository can only be Buildrequest, and the messages will be enqueued in the C&R que. The command of the message from the Child Builder is get build request, this means that the Child Builder is unoccupied, the message will be enqueued in the C que. If the C&R que and the C que are both not empty, then the Mother Builder will dequeue the message to get the ready Child Builder and the message. If the message is close, close the ready Child Builder, else, send the Build Request to the Child Builder, the Mother Builder will send a message to tell the Child Builder which Build Request to parse after send the Build Request.

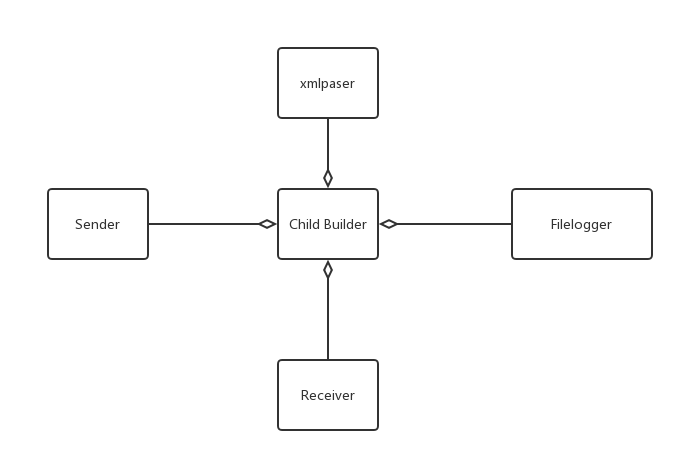
1. Here the Child Builder will start the build work after it receive the message with the command and the name of the Build Request.
2. When we create the host of the communication services, we can get and send messages and files from other remote servers.

### 4.5.2Child Builder

The Child Builder uses the MSBuild to build the files into the libraries.

4.5.2(a)structure

Here I use the class diagram of the Child Builder to show the structure of the Child Builder

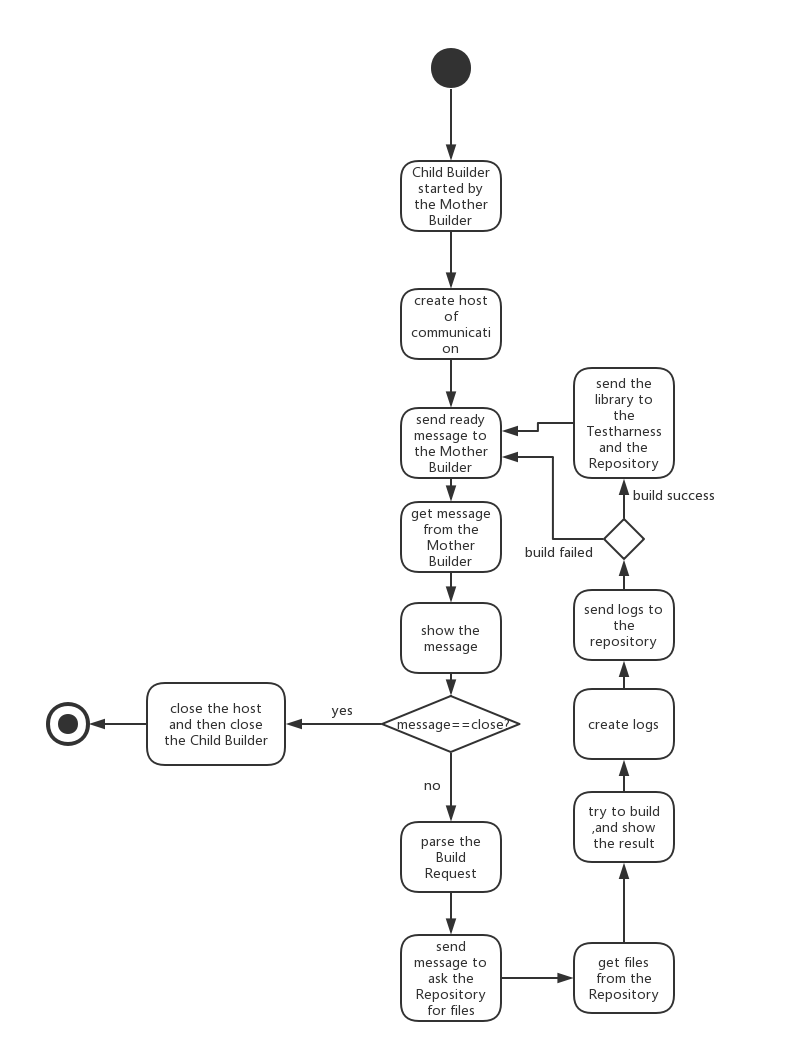


*Figure 10 Class Diagram of the Child Builder*

The Child Builder has one Receiver and three Sender, the Receiver get messages and files(the message tell the name of the Build Request, and the file is the Build Request) from the Mother Builder and get files from the Repository(the files need to be built). The first Sender will send the message to tell the Mother Builder that itself is ready for new Build Request or close, the second Sender will send logs and the libraries to the Repository, the third Sender will send the libraries to the Testharness, and after send the libraries to the Testharness, this Sender will send a Test Request(a message tell the Testharness which library to test). In my design, the Testharness can test the library if it knows the name of the library, and there is no requirement tell us to use the xml file pass the Test Request, so I only use a message to make things easy. The Filelogger is a class from the Microsoft, it can log the Build logs to the txt. Xml parser will parse the Build Request.

4.5.2(b)activities

The activity diagram shows the activities of the Child Builder



*Figure 11 Activity Diagram of the Child Builder*

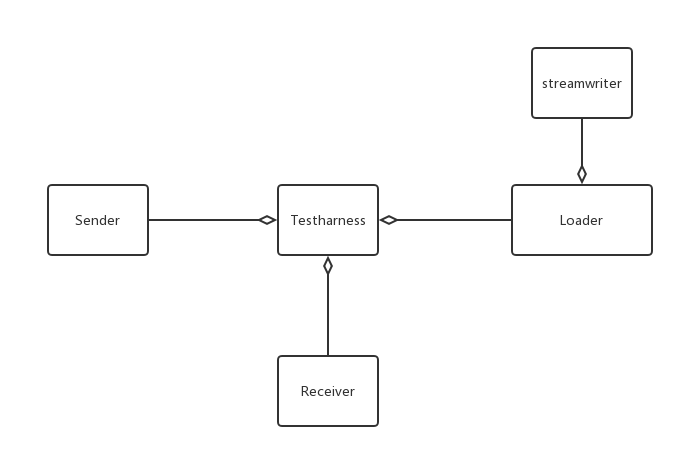
When the Child Builder is started, it will create a host of communication service and then send a ready message to the Mother Builder. When receive a Message from the Mother Builder, The Child Builder will show the message in the console. If the message’s command is close, the Child Builder will close the host and then close itself. Else it will find the Build Request(the xml file in its storage) according to the message and then parse the Build Request. The Child Builder will get the files from the Repository according to the message in the Build Request(send a message, the command of the message is getrepofiles, and the Repository will send the files to it), try to build it and show the result in the console. If build failed, the Child Builder will create build log, send it to the Repository, and send a ready message to the Mother Builder. Else if build success, the Child Builder will send the log and the library to the Repository, send the library to the Testharness, and then send a message(test request) to it. After doing all these things, the Child Builder will send a ready message to the Mother Builder.

## 4.6Testharness

The Testharness test the libraries being built by the Build Server.

### 4.6.1 structure

The class diagram shows the structure of the Testharness.

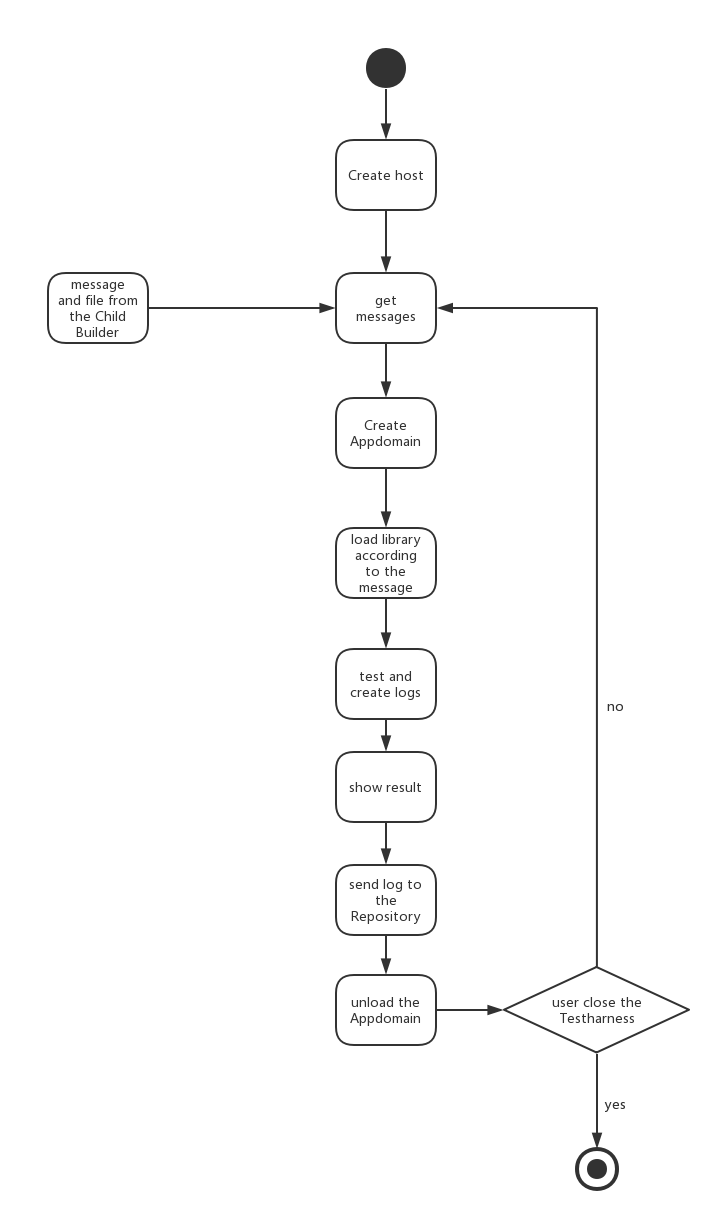


*Figure 12 Class Diagram of the Testharness*

The Testharness has one Receiver and one sender, the receiver will get the libraries and the test requests from the Child Builder. The sender will send the test logs to the Repository. The Loader will load and test the library. The stream writer will write test logs to the txt files.

### 4.6.2activities

The activity diagrams shows the activities of the Testharness.



*Figure 13 Activity Diagram of the Testharness*

When the Testharness runs, it will create a Receiver to receive messages and files, when the Testharness get the message from the Child Builder, it will create an appdomain, load and test the library in the appdomain, then show the result in the console and send the log to the Repository. After doing all these things, the Testharness will unload the appdomain.

# 6.Critical Issues

## I never write any code before I come here, how can I implement the project before the deadline

How to write code, how to learn C# quickly, and implement the project before the deadline.

**Solution:**

* Read the C# 6.0 in a Nutshell
* Always go to Dr. Fawcett’s and TA’s office hour
* Looking at Dr. Fawcett’s code

## Running Steadily

There are a lot of parts in the system, it is really easy to crash while running.

**Solution:**

* Test the whole system again and again and fix the bugs.
* Always using the try{} catch{} keywords.
* Using process pool, even one of the Child Builder crash, the other Child Builder is still work, and the whole system would not crash.

## 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?

**Solution:**

When a server receives or sends a message, the server should show that message on the console, when send files, show the source and destination path, so that the user can see what the system doing clearly, the build and test result should also be demonstrated to the console, something like “build success”, but the details can be demonstrated in the log files. When the server send a file or create a file, it should show the store path in the console.

## 5.Thread safe

If the code is not thread safe, it will create a lot of problem, such as dead lock.

**Solution:**

Using Blocking queue when passing messages, and the every action of the server is cause by a message, this will make the system thread safe.

## 6.The Repository sometimes will crash if I select several Build Requests to build at the same time

**Solution:**

I do not have enough time to fix it , so I just let the thread sleep, after send each request.

## 7.Performance

We may have a lot of files to build how to make the build work efficiency

**Solution:**

Using process pool to start several Child Builder, and the each Child Builder work at the same time, doing the different build work at the same time will makes the system efficiency. the Mother Builder will send the Build Request to the ready Child Builder.

## 8.When the Child Builder receive the Build Request, the Child Builder will parse it and send file request to the Repository, then build the files. But when the Child Builder start to build, some needed files maybe not received.

**Solution:**

1.This is the most easily way to solve the problem, just let the Child Builder’s thread sleep for a while, after send the file request to the Repository, I choose this way because I do not have enough time.

2.After Child Builder send all file requests to the Repository, send a message, the message with some special information, such as “end of the file list” or other things, when the Repository deque this message, it means that all the files have been send to the Child Builder, and then the Repository will send a message to the Child Builder, the Child Builder will build the files into library after receive this message.

# 7.Conclusion

Each Server in the system uses WCF to communicate with others, so they are remote, and they can run on different machines. Communication is the most important thing of the whole system, because the actions of the servers are driven by the message.

Although we design other servers in the system, the Build Server is the server we focus on, we use the process pool to make it works efficiently, and safety, use MSBuild to build the files into libraries, and use the WCF to send files and messages, get files and messages.

# Appendixes

## 1.1The structure of the CommMessage

The CommMessage is really important in the system, Because every action of the system(except some action of the GUI will be controlled by the user) is caused by the message.

The CommMessage include the following part:

1. source, where this message is from.
2. destination, where this message is to.
3. Body, the body always include the important information of the message, such as filenames.
4. Command, the server will do the work according to the command, such as send files or build.

## 1.2Changes and Deficiencies

1.In my project#1, I have a wrong idea about the whole system, I thinks that several users will use the system at the same time, though it can be implemented, but it is really hard for me now. I change the design, the system just has one user one time.

2.Some of the diagrams are not correct in my project #1, for example, do not use the Synchronizing bars correctly. I redraw all the diagrams of in the OCD based on my code for project #4.

3.In the project#1 I do not consider about the communication, all the servers are local server, just like what we do in the project #2, one package call another package. In the project #4, I used the WCF to let each part communicate, so that each server is remote.

4.In my project#1’s design, if the build work cause the Build Server crash, the whole system will crash, in my project#4 design, using the process pool, even one of the Child Builder crash, the whole system will still work.

5.The Critical Issues for Project one are something I get on the internet, and for the project #4, are all the things from my project.

# Reference

1. midterm exam
2. https://ecs.syr.edu/faculty/fawcett/handouts/webpages/CSE681.htm