**CSE 681 – SOFTWARE MODELING AND ANALYSIS**

**OPERATIONAL CONCEPT DOCUMENT**

**REMOTE BUILD SERVER**

**Instructor – Prof. Jim Fawcett**

By -

Chetali Mahore

SUID - 750500177

6th December 2017

**Contents**

[1. Foreword 3](#_Toc500362243)

[2. Executive Summary 4](#_Toc500362244)

[3. Introduction 5](#_Toc500362245)

[3.1 Objective and Key Idea 5](#_Toc500362246)

[3.2 Obligations 5](#_Toc500362247)

[3.3 Organizing Principles 5](#_Toc500362248)

[4. Users And Use Cases 6](#_Toc500362249)

[4.1 Users and Use Cases 6](#_Toc500362250)

[4.2 Future scope 6](#_Toc500362251)

[5. Application Activities 8](#_Toc500362252)

[5.1 Package Diagram 8](#_Toc500362253)

[Entire System 8](#_Toc500362254)

[1. Executive 8](#_Toc500362255)

[2. GUI 8](#_Toc500362256)

[3. Mock Repository 8](#_Toc500362257)

[4. Builder Server 8](#_Toc500362258)

[5. Child Builder 8](#_Toc500362259)

[6. Mock Test Harness 8](#_Toc500362260)

[Build Server 9](#_Toc500362261)

[5.2 Activity Diagram 10](#_Toc500362262)

[Entire 10](#_Toc500362263)

[1. GUI 11](#_Toc500362264)

[2. Mock Repository 11](#_Toc500362265)

[3. Builder 12](#_Toc500362266)

[4. Test Harness Mock 13](#_Toc500362267)

[5.3 Class Diagram 13](#_Toc500362268)

[Build Server 13](#_Toc500362269)

[5.4 Comm 15](#_Toc500362270)

[6. Comparison of OCD 1 and Design Document 16](#_Toc500362271)

[7. Issues 17](#_Toc500362272)

[1. Performance 17](#_Toc500362273)

[2. Concurrent Access to Repository Files 17](#_Toc500362274)

[3. Usability 17](#_Toc500362275)

[4. Complexity 17](#_Toc500362276)

[5. Flexibility 17](#_Toc500362277)

[8. Deficiency 18](#_Toc500362278)

[9. Conclusion 19](#_Toc500362279)

[9. Appendix 20](#_Toc500362280)

[10. References 21](#_Toc500362281)

# Foreword

Operational Concept Document(OCD) is the foremost step in understanding the working of large systems. We are designing an OCD for a Remote Build Server. The goal of our project is to develop a Remote Build Server which interacts with other components like Client Mock, Repository Mock and Test Harness Mock.

The objective of building remote build server is to maximise the performance and usability of the system.

Our project focusses towards developing a local build server, but our final task is to design remote build server. Since the build server would be placed at a distance from other modules, there arises a need of communication framework. The message and control passing between the modules is handled by Windows Communication Foundation which is implemented as a part of Project 3.

The four projects are as follows:

1. Operational Concept Document for Build Server
2. Core Build Server
3. Remote Build Server Prototypes
4. Remote Build Server

# Executive Summary

As referenced from the Mid Term Paper 1 Solution, the executive summary is as follows:

This development will create a Build Server, capable of building C# libraries, using a 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 a communication channel and 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 Builders, and a WPF client that supports creation of build request messages.

Finally, the third stage, Project #4, completes the build server, which communicates with 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.

The various modules of the system are: Client Mock, Repository Mock, Remote Build Server and Test Harness Mock.

The Client Mock will generate the test request which will be stored in the Repository. The request is then passed to the build server whose function is to build it. To maximise the performance of the system, we created child build servers which receive the request and parse it. Parsing helps in determining the test dependant files which are loaded from the repository. The child build servers then build the request and create a .dll file and sent to the test harness for execution. The build logs are then sent to the repository. The Test Harness Mock executes the .dll file and returns the result as passed or failed to the Client and sends the test log to the Repository.

Critical issues include: Concurrent access to repository files, building source code using more than one language, scaling the build process for high volume of build requests, and using a single message structure for all message conversations between clients and servers. All of these issues have viable solutions.

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

The software development process is a comprehensive process used for designing and implementation of the software product.

Requirement gathering and analysis is the first and foremost step towards development of the software. Questions like who are the users, what is the nature of the problem, how will it affect the users, what could be the possible solution, is the solution feasible, should be answered. Once the answers to these questions are found, the system analysis is carried out. The solution is judged if it is realistic and in accordance to the current technology. If yes, we proceed towards designing of the product. The design consists of architectural, high-level and low-level design.

The design is now implemented in the form of packages, classes and methods. Testing of the product is done rigorously before it is deployed in the market.

For developing a build server, we are following the waterfall model of software development. We have clearly defined the requirements and the product definition is going to be stable over a period. The waterfall model is a linear model of development.

The system is composed of four modules: Client Mock, Repository Mock, Remote Build Server and Test Harness Mock. The architecture overview of the system is the client is requesting the test requests to be executed. To execute the test requests, the files and dependencies are to be loaded from the repository and they should be built by the build server. The built files are then sent to the Test Harness for execution.

## Objective and Key Idea

The objective of this project is to maximise the performance of the remote build server by creating child builders which build .dll files for the test harness to execute. The client sends a build request to the remote build server. Depending on the availability of the child builders, the mother builder allocates it the build request. The function of the child builders is to parse the request, fetch the required dependant files from the repository and build the request. If the building of the request is successful, its .dll file is created and sent to test harness for execution.

## Obligations

The primary obligations of the remote build server are as follows:

* Effective communication with the components of the system via Windows Communication Foundation.
* Accepting the build requests created or sent by the client process and allocate them to child builders for maximising the throughput and performance of the system.
* Sending the build files to the test harness for execution.

## Organizing Principles

* The components of the system i.e. Client Mock, Repository Mock, Remote Build Server and Test Harness Mock are placed in independent packages for special functionality.
* These components communicate with each other only through Windows Communication Foundation
* Each of the module has its own local storage for storing the files and processing them.
* There is asynchronous message passing between the modules, thus making the entire system asynchronous.

# Users and Use Cases

## Users and Use Cases

The users of the system are the entities which interact with the system. The users can be classified into four types – developers, managers, quality assurance and testers

**Developers:**

Developers are responsible for the design and implementation of the software product. They design different modules of the software and perform unit and logical testing on the developed code.

**Uses:** They expect the system to be agile enough to run various build requests simultaneously and to get accurate results. The Graphical User Interface (GUI) for the developer would be focused towards improving the front-end design, creating new build requests, loading the files from the repository for displaying it onto GUI. In addition, the developer will also receive updates and notifications from build server and test harness about the processing of build requests.

**Managers:**

Managers have a key responsibility of doing requirement gathering and analysis of the project. The managers lead the project group. The manager lays down the guidelines for the project and monitors it.

**Uses:** Running the build requests and deriving the results is the primary job of manager. User interface for a manager would allow him/her to run and schedule the processing of test requests, for example, schedule the test requests at night and check the test logs in the morning. The manager should have a quicker access to the various versions of the log files and the metadata stored in the repository. This information can be used to analyse the data patterns and test results.

**Quality Assurance:**

The quality assurance team focuses on processes and the entire flow of the software. It is responsible for maintaining the quality standards within the organization and with the outside world.

**Uses**: The QA team expects the software to meet the requirements as specified by the users during requirement gathering. The design of the user interface for the QA team should simplify their task of mapping the requirements to the product developed. Running the executive program, which displays the requirements met would ease the job of the QA team.

**Testers:**

The tester’s job is to create and execute various types of test cases on the developed product.

**Uses:** Testers help in refining the software system. The bugs and errors are found out at an early stage of development which increases the reliability of the system. The tester expects the system to run multiple build requests with maximum accuracy. Load bearing capacity of the software is also tested. The tester should be able to append the existing build request and generate new build requests too. The interface of the tester should be devised to make testing process easy.

## 4.2 Future scope

The main scope of the project is to develop a build server which interacts with modules such as Client Mock, Repository Mock and Test Harness Mock.

Our immediate goal is to build the server locally on the same machine as that of other modules. The build server interacts with other modules via Windows Communication Foundation. Our ultimate purpose is to design a remote build server which would interact with other modules of the system remotely. The implementation of local build server is as below:

1. The modules of the system are encapsulated in different packages.
2. Each module has communication prototype implemented in it. i.e. each module has its own receiver which is started at a specific port.
3. Different processes are started for different modules and the interaction between them is controlled by the exchange of messages.
4. Initially, the user inputs the number of child processes to be spawned. Accordingly, the mother builder allocates different receiving ports for each of the child processes.
5. The client initiates the flow of messages by creating the build request and sending it to the mother builder.
6. Depending on the availability of the child processes and the number of build request received, the mother builder allocates build request to each of the child process.
7. The function of the child process is to parse the build request and find the list of dependent files from the build request. Each of the child process has its own local storage for storing the files received from the repository. If the dependent file is present in its own storage, the child uses the same file. If the file is not found in its storage, the child requests the repository to transfer the file.
8. After successful transfer of the file, the child builder attempts to build the build request. If successful, a .dll file is created which is sent to the test harness. A log of successful or failure of build request is generated which is sent back to the repository for storage.
9. The Test harness executes the .dll file and displays the result on the its console. The results are even displayed on the client console and the test log is sent to the repository for future use.

Features of the remote build server:

1. **Accessibility:**

The server can be accessed from any location since the socket connection takes care of the IP addressing.

1. **Robustness**

The build server is capable enough to handle multiple build requests at the same time. Each build request is sent to each of the available child builder and thus, the server performs multiprocessing.

1. **Reusability:**

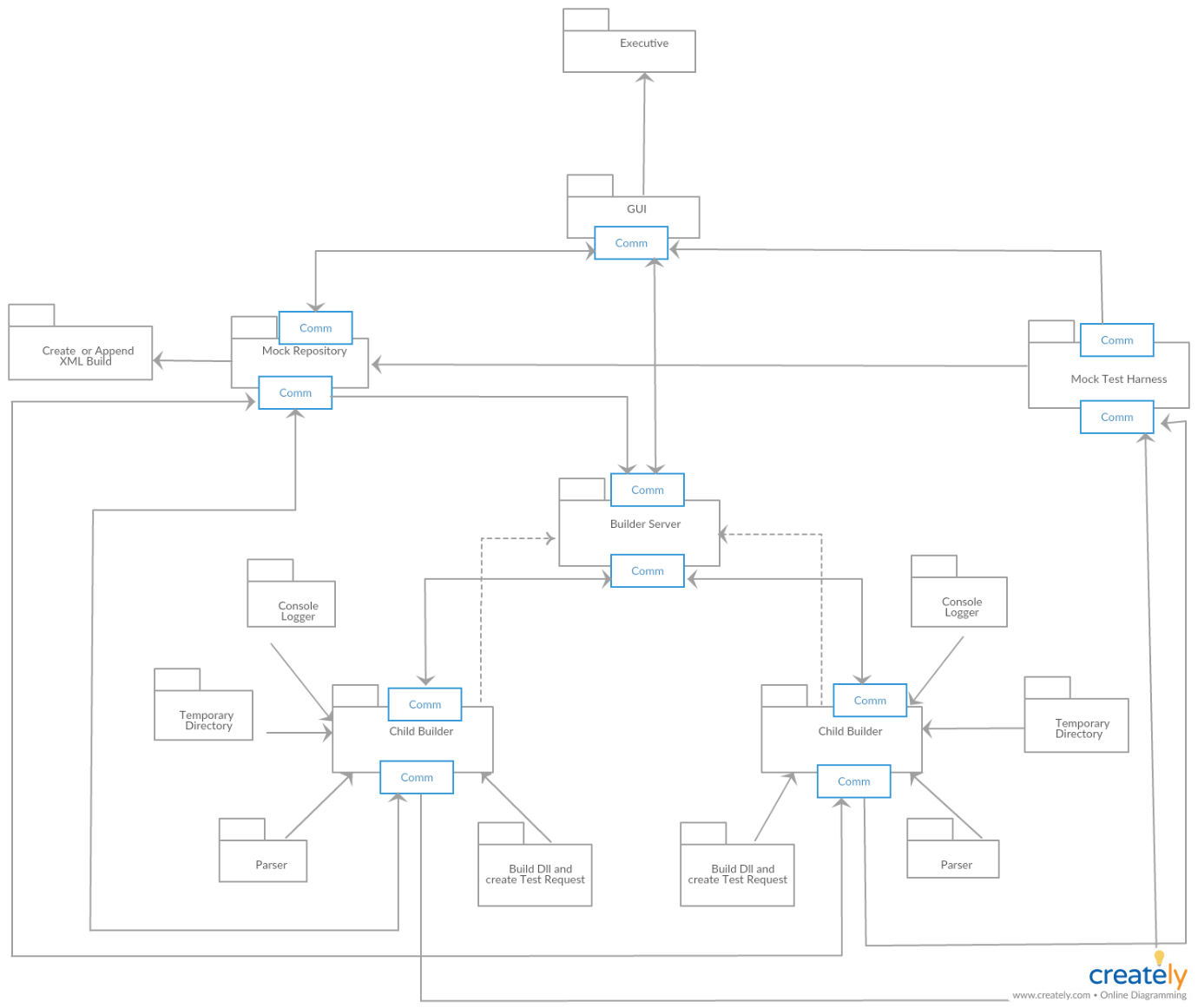
The build server can be developed as a standalone module interacting with various other modules. The build server can then be integrated into other software systems where building of requests is to be automated.

The message passing and the communication between the modules will be handled by the Windows Communication Foundation Framework.

# 5. Application Activities

## 5.1 Package Diagram

### Entire System



The package diagram has six main components. They are as follows:

### 1. Executive

### 2. GUI

### 3. Mock Repository

### 4. Builder Server

### 5. Child Builder

### 6. Mock Test Harness

Executive is the start-up module for this system. It has a control of all the message passing implemented in the system. On running the executive, the users, developers, testers as well as managers can easily identify the requirements satisfied by the project.

GUI is the user interface for the system. It allows the user to spawn its own number of child processes, access the files from the repository, build or append the XML build request and send the XML build request to the mother builder. The results of building the build request are displayed on the GUI which allows the user to check the build logs. The results of the execution of the test harness is displayed on the GUI.

The Mock Repository accepts the inputs of testdriver, test files from the GUI while creating a build request and sends the created XML request back to GUI. The repository is even responsible for appending the existing build request. Upon request from the child builders, the repository transfers the files to the child builders. After attempting to build the request, the child sends back the build logs to the repository. On execution of the created test request, the test harness sends back the test logs to repository for storage.

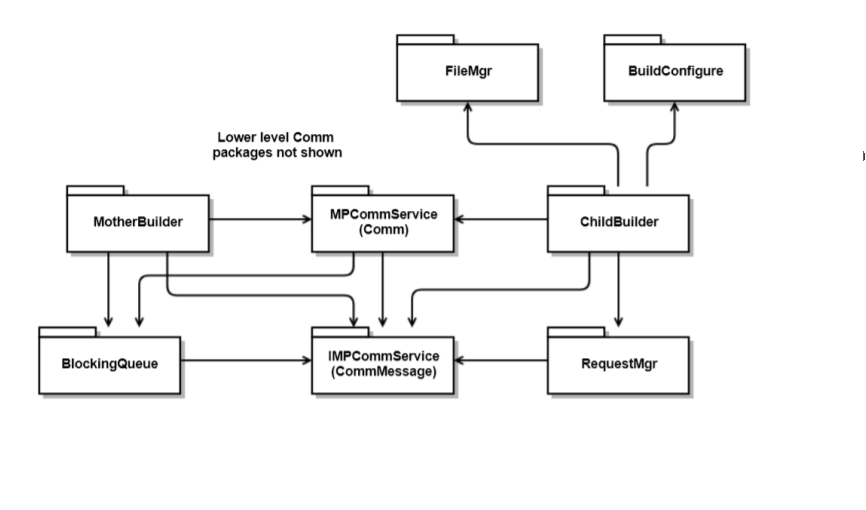
Builder Server is responsible for spawning the number of child processes which is received as an input from the user. The Build Server accepts the XML build requests from the repository and allocates them to the available child process. When the child completes the processing, it sends the ready message to the builder server. Thus, the function of the builder server is monitoring the child processes status and receive XML build request from the repository. If a quit message is sent from the GUI, the builder server shuts down its pool processes.

The primary function of the child builder is to parse the build request received from the builder server, fetch the dependant files from the repository and attempt to build the build request. It then sends back the build log to repository. Moreover, if the build request is successfully built, its test request is created with the .dll files in it. These files are transferred to the test harness for execution.

Mock Test Harness accepts the test request file and tries to execute it. It sends back the test log to the repository.

### Build Server

From Mid Term solution Paper 4 by Prof . Fawcett, the package diagram for build server is as follows:



Explanation from Mid term solution paper 4,

MotherBuilder manages ChildBuilder processes

ChildBuilder builds test libraries and sends to TestHarness

BuildConfigure sets the environment and provides access to build tool chain

FileMgr creates and deletes temporary build directories and provides access to files in those directories.

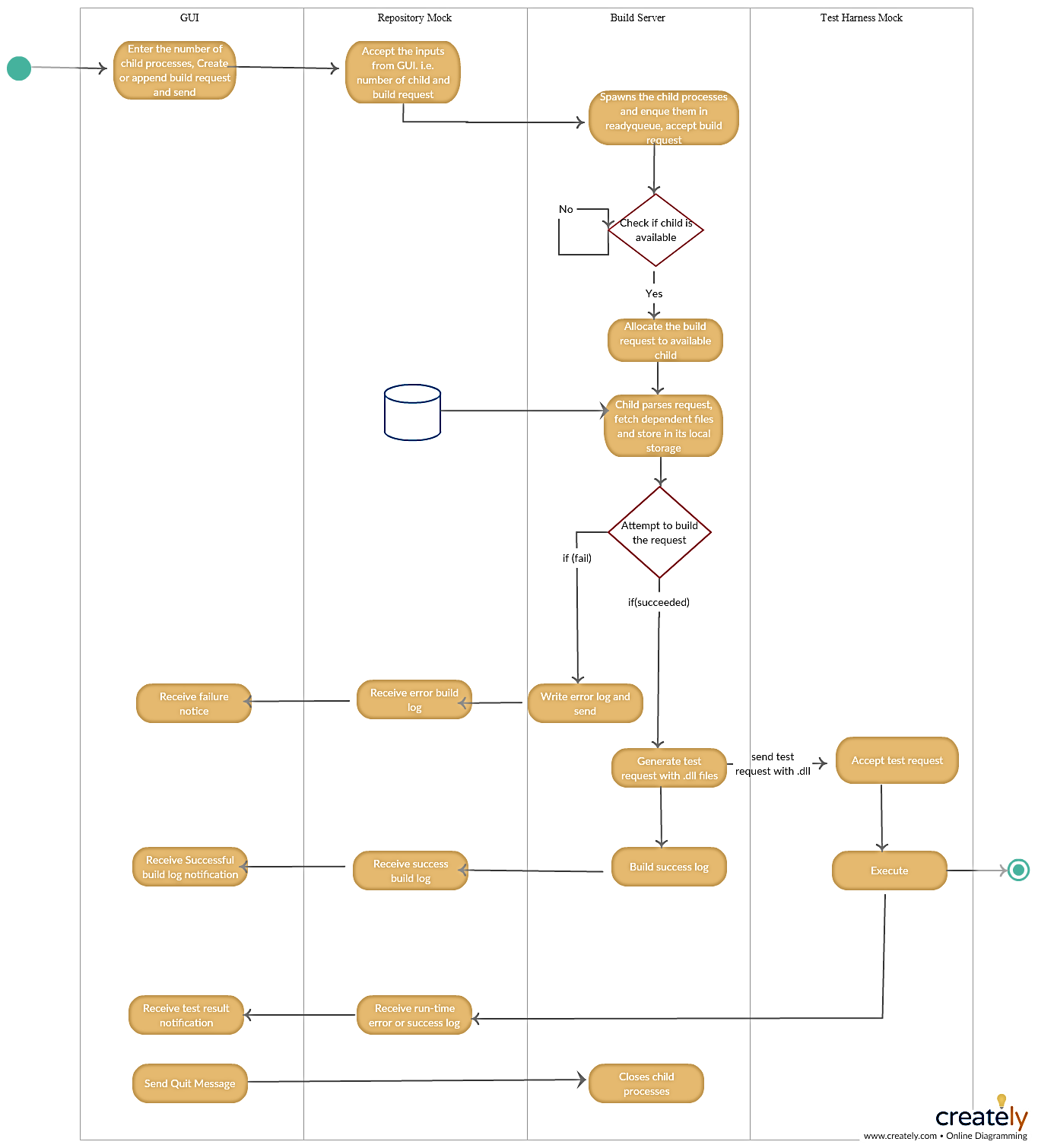
RequestMgr parses BuildRequest messages and creates TestRequest messages

MPCommService provides communication services used by MotherBuilder and ChildBuilders

IMPCommService defines the CommMessage structure used by communicators

## 5.2 Activity Diagram

### Entire



The figure above is the overview of the entire system. The activity diagram shows the interaction between the modules. The main components of the proposed system are as follows:

1. GUI
2. Repository Mock
3. Build Server and child builder
4. Test Harness Mock

### 1. GUI

The GUI is the actor of the build server system. The client is responsible for interacting with the external world and with the internal components.

Tasks performed by client mock:

1. Create Test Request

A test request is an xml file which contains elements like test, driver and tested. The test element contains test driver and tested sub-elements. The user is given the flexibility to select the files and send to repository for building the test request. The user can run new tests along with some old ones to check the reliability of the system.

1. Accept from Repository

The GUI has the updated XML file shown.

1. Command Repository to process test request

The GUI commands the repository to process the request at a specific time. The client may ask the repository to start processing the test request at midnight and display the results in the morning when the client logs in again.

1. Accept build logs and view

It is the job of the GUI to display the build logs generated by the build server and sent by the repository. The build server displays compile time errors and warnings to the client. If the files are compiled and built, the build server sends a success log to the repository which in turn sends to GUI. However, if the files are not properly compiled or the files may not be coherent to the test input data, the build server sends error logs to repository.

Example: Syntax errors, Typechecking errors

1. Accept test logs and view

The build server sends the compiled files to test harness for execution. The results of the test cases processed by the test harness is also sent to client for its review.

During the processing of the test requests, the client mock may receive notifications about runtime errors like deferencing a null pointer, division by zero, running out of memory. It is the duty of the client to handle these notifications and take immediate action.

### 2. Mock Repository

The repository mock is the reactor in the system. The repository mock’s main function is to accept the test request from the client and provide the client with necessary log information. The repository mock stores the log files of various versions. It also manages the metadata. A repository is a semi-automated storage mechanism that provides pluggable policies for:

* File management
* Version control
* Package ownership
* Checking and Checkout
* Package browsing

Below are the tasks performed by the repository mock:

1. Accept the files and build the Request

2. On-command

Following are the steps performed after the on command.

1. Send the Build Request to Build Server

The build Request contains the test drivers and the files which are to be tested.

1. Accept build log

Upon request from the child builders, the repository transfers the files to the child builders. After attempting to build the request, the child sends back the build logs to the repository.

1. Accept test log

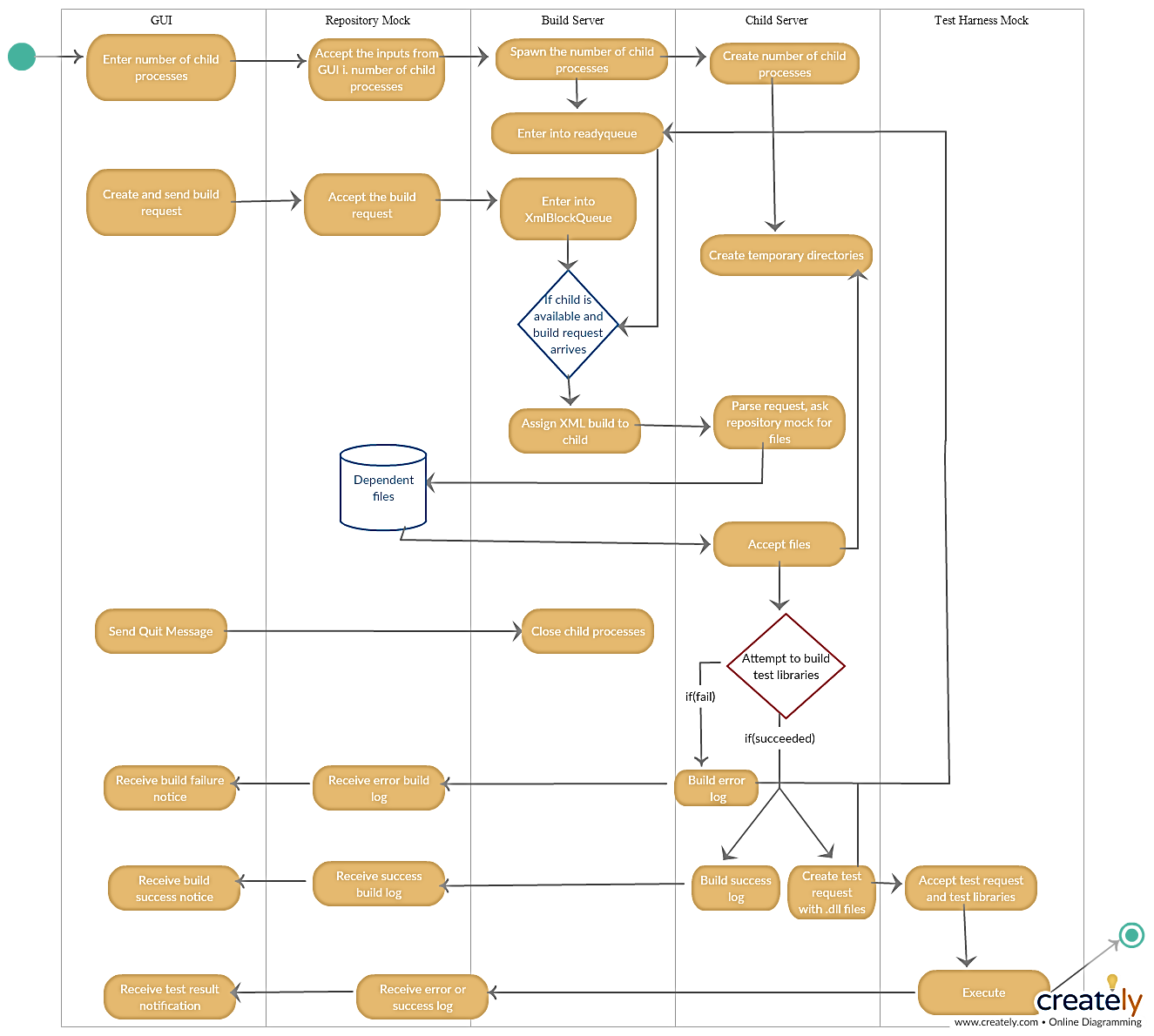
The built test libraries are sent to the Test Harness for execution. If the execution of the test cases is successful, a log stating the same is sent to the repository. If there were any runtime exceptions encountered during the process, an error log is reported to the repository. The repository stores the test results and displays them to the client when he/she asks for it.

1. Send build log to client and send test log

The repository is responsible for logging and metadata management. The results of building the test libraries and execution of the test cases are stored in the repository. The repository makes these results available to the client when the client demands for it. The logging of data is important for two purposes:

1. If any of the module i.e. client, build server, collaboration server or test harness crashes, the logs are still safe. The data can be rolled back to a previous breakpoint to recover the lost data. A clone of the repository should also be made on a timely basis for precaution against crash.
2. The logged data could be utilised for analysing the data patterns. Strategic business solutions can be obtained by proper analysis of data.

### 3. Builder



The build server plays the role of a reactor in the system. The build server takes the test request and test files from the repository and builds test libraries.

Following are the steps involved:

1. Accept the number of child processes

Spawns the number of child processes, creates their temporary directory, inserts the children in readyqueue.

1. Accept XML build request

It accepts the build request from the repository and allocates it to the available child.

1. Child process accepts the build request

Child parses the request, fetches those files from the repository and stores in its temporary storage. Attempts to build the request. If successful, test request with the .dll library files is made and sent to the test harness.

1. Sends the logs to the repository

It sends back the build logs to the repository.

1. Signals the mother that it is ready

It sends back the message to mother that it has finished its work and is ready to process another build request.

1. Closes child processes

Upon request from the GUI, the mother shuts down its child processes.

### 4. Test Harness Mock

The test harness mock is the reactor of the system which runs test cases concurrently for multiple users. The test harness executes tests, logs results and submit results to the repository. It also notifies the author of the tests of the results.

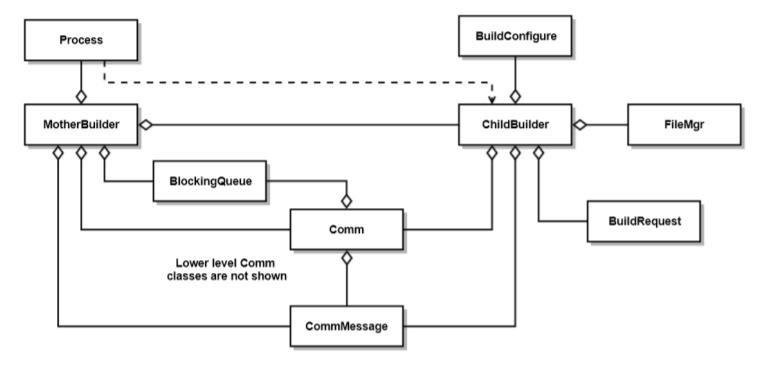
The steps are as follows:

1. Accept the test request and the libraries from child builders
2. Start the execution of these files.
3. If (unhandled runtime exceptions)
4. Stop the execution, send the error log to the repository and a notification to the client.
5. Else send the success log to the repository and a prompt to the client for successful execution.

## 5.3 Class Diagram

### Build Server

From Mid term solution of paper 3 by Prof. Fawcett,



- MotherBuilder manages ChildBuilders, passing them BuildRequests when ready

- ChildBuilder loads files, matching BuildRequest, from Repository and builds them into libraries and sends them to the TestHarness

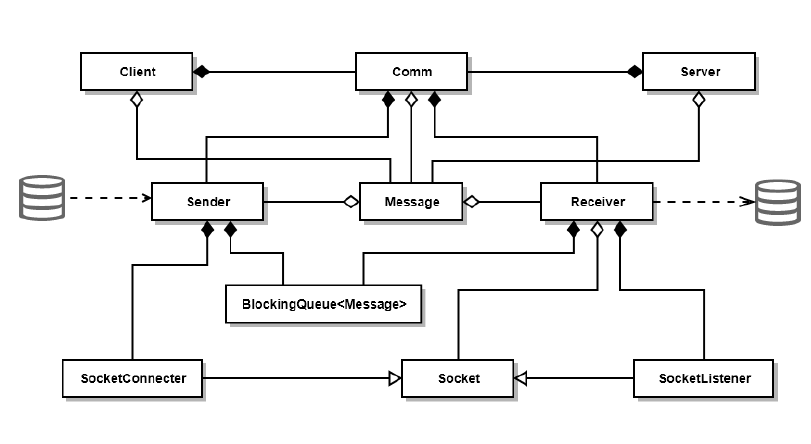
- BuildConfigure sets the environment and paths to toolchain to prepare for build

- FileMgr is used to create temp directory, find the libraries for Comm.sendFile, and delete directory.

- System.Diagnostics.Process is used to support starting childBuilders and tracking their exit events. - Comm and CommMessage support communication between MotherBuilder, ChildBuilders, Repo, and TestHarness

- BuildRequest parses BuildRequest message for source code file names

## 5.4 Comm



Comm which consists of Message, Client and Server. The Sender and receiver are the main classes of the comm prototype which helps in connection and communication between the different modules of the system. BlockingQueue is implemented in each receiver and sender classes which controls the message flow into the system. It avoids deadlock and thread interference.

# 6. Comparison of OCD 1 and Design Document

OCD 1 gave an overview of the entire project. The system was implemented in project 2, 3 and 4 which made the idea of the entire system clearer. The design document entails the implementation features and the design of the entire system. The design document enables the developers and the managers to understand the message passing communication between the various components of the system. The design document also considers the design constraints, the critical issues and the deficiency of the project. It enables to have a better and clear picture of the entire system.

# 7. Issues

## 1. Performance

**Issue**: To maximise the throughput of the system, the build server spawns a number of child processes. Each of the child process attempts to build the build request and if successful, creates a test request with the .dll files present in it. The test requests are then sent to test harness for execution. Since, there is just one Mock Test Harness, the load on the Test Harness will increase. Suppose there are 100 build requests generated from one user, there are millions of users generating build requests, the test harness will be overloaded with work.

**Solution:** There must be a buffer mechanism implemented in the build server and the test harness. If the buffer size of the test harness is reaching its threshold, it should warn the build server about it and the build server should stop sending the test requests.

## 2. Concurrent Access to Repository Files

**Issue**: There is a much possibility that many users want to execute the same XML build request which leads to concurrent access to files present in the repository.

**Solution**: Introduce synchronization on the program block which is accessed simultaneously by user threads. Insert try – catch blocks which handles the exception condition and allows the process to get the read-write access to the file.

## 3. Usability

**Issue**: The GUI designed for users should be easy to use. If the GUI is not user friendly, there will be customer dissatisfaction which would result in a loss for the company.

**Solution**: The GUI designer should perform thorough testing on the front-end design during development. The designer should use GUI for all his/her work. This will lower the risk when the software product is released in the market.

## 4. Complexity

**Issue**: The system is a multi-threaded asynchronous system. Each of the module has its own receiver thread running for receiving the incoming messages from the other modules. Moreover, the mother builder or the builder server spawns a number of child processes for multi-processing of build requests. Each of the child process has its own receiver thread running. There is a possibility that the threads may interfere.

**Solution**: The Message passing communication with implementation of BlockingQueue using windows communication framework helps in synchronization of tasks. Thread interference is avoided.

## 5. Flexibility

**Issue**: How to define a single message structure that works for all messages used in the Federation.

**Solution**: A message that contains To and From addresses, Command string or enumeration, List of strings to hold file names, and a string body to hold logs will suffice for all needed operations.

# 8. Deficiency

Here are some of the following deficiencies of the project:

* The project doesn’t include a tool checker. Only C# files are used for creating, appending and executing the XML build request. The project doesn’t have a check for other types of files like C++ and Java.
* The Test harness executes the .dll file in the same App domain. The child app domains are not created for executing the .dll file. AppDomain Manager

9. Conclusion:

The objective of the project ‘Remote Build Server’ was to build a remote build server which will communicate with other modules of the system remotely.

The **Operational Concept Document, Project 1** was a comprehensive document which helped me to analyse the uses, structure, activities and issues involved in developing a software product.

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 a communication channel and process pool.

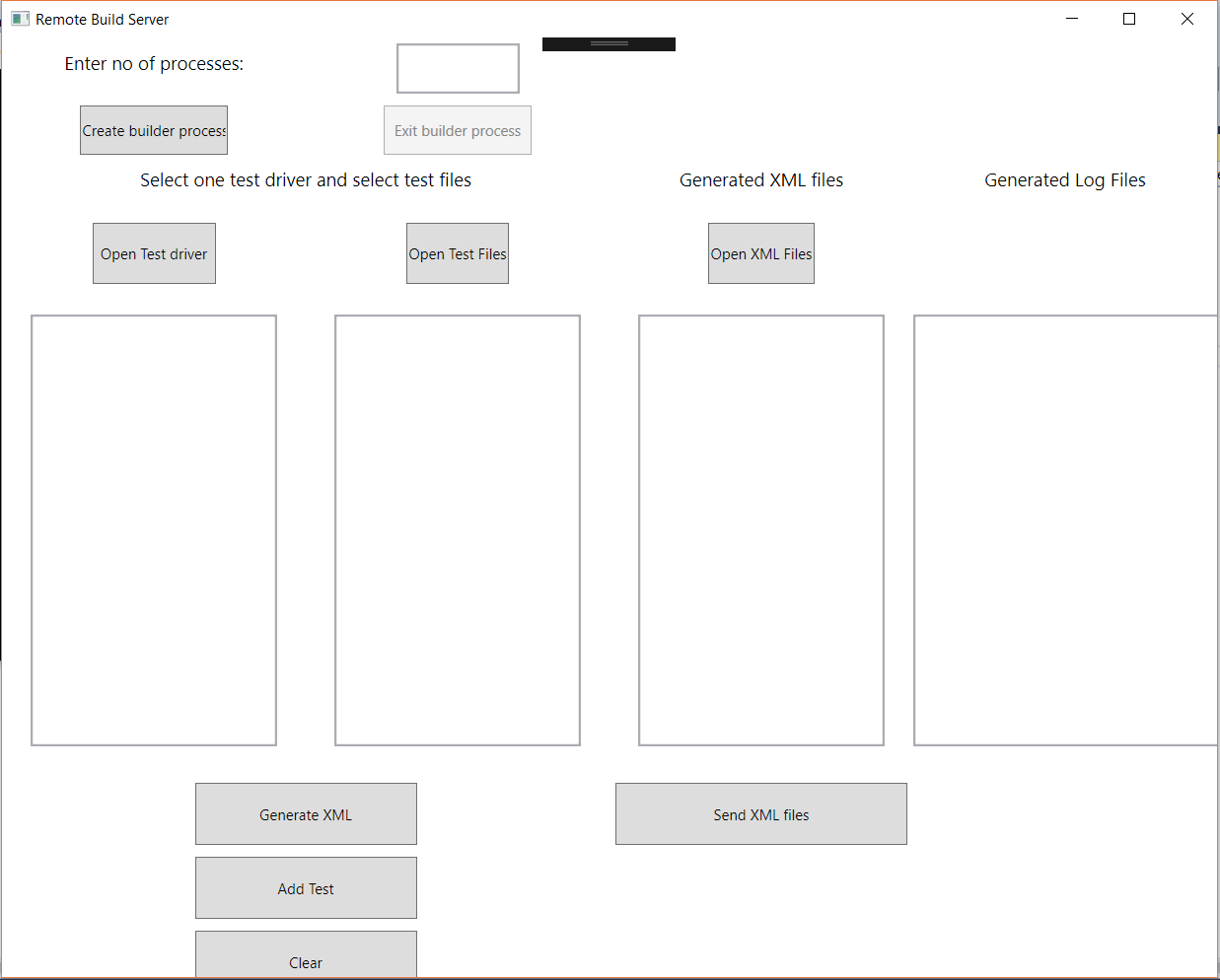
The **second, Project #3,** develops prototypes for a message-passing communication channel, a process pool, that uses the channel to communicate between child and parent Builders, and a WPF client that supports creation of build request messages.

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

These projects helped me to understand the windows presentation framework and windows communication framework. These projects helped me to enhance my logical and technical skills. The message passing in asynchronous system was best understood through the implementation of these projects. It was a great opportunity to work under Prof. Fawcett and complete these challenging yet interesting projects.

# 9. Appendix

Here is the screenshot of the GUI window:



# 10. References

1. <https://ecs.syr.edu/faculty/fawcett/handouts/webpages/cse681.htm>