BUILD SERVER WITH CONTINUOUS INTEGRATION

OPERATIONAL CONCEPT DOCUMENT

CSE681- Project 4

By Dinesh Kumar Dhamotharan (SUID: 586563818)

Instructor: Jim Fawcett

Date:12/07/2017

Contents

Figures

1 EXECUTIVE SUMMARY

Developers typically work on particular parts of an assignment in the large scale projects, which must be systematically merged into one whole of the project while maintaining the integrity of existing work. The integration is difficult which is due to the potential of multiple developers working on the same files of the code base on their distinct copies. To resolve this kind of integration issues and to automate the testing for the newly integrated files, the build server which supports continuous integration and test automation can be used.

The purpose of the project is to help the development and testing team by integrating and building the code continuously and by automating the testing process. The project should provide the following capabilities:

- Allow the users to add/update or remove the code and documents in the repository.
- Allow the user to create and process the test request by sending it to build server.
- By the test request, the build server must take the build automatically and must be stored in the repository. The successful build process must send the libraries to the test harness and the testing process must be triggered automatically.
- On receiving the files from build server, the test harness must start executing the test cases. On successful completion of testing process, the test harness must send a notification to client.
- Logs must be captured during both build and testing process and these files must be stored in repository server for resolving the occurred problems.

The primary users of this tool will be developers, project leads/managers, test team, Quality assurance team and the customer. Developers could use this toolset for streamlining development, integrating code into shared repository and get the rapid feedback of the feasibility of that code. The health of the code will be notified to stakeholders like customers, QA and Managers on every changes and so the developers get the feedback as fast as possible.

Below are some of the critical issues associated with the project, and the solutions are discussed in the later sections.

- Authorization and Authentication: The system will be vulnerable to data leaking if it doesn’t verify the person is specific user of the system.
- Version Control: Any data that checked in the repository must be versioned for organizing them.
- Inconsistent input data: While sending the test request, the user can create a xml file with his own schema.

This document further provides discussions on flow of activities for all tasks, interactions between the modules of the application, solutions for the critical issues and the use cases associated with the application.

2 INTRODUCTION

2.1 Objective and Key Idea

The primary objective of the application is lessen the risks of integration and to automate the build and testing process by continuous integration such that it can offer higher performance by building stuff faster and decreasing the code review time. It will increase the productivity of stakeholders and helps the development team to integrate the developed code instantly and to get the feedback as fast as possible.

2.2 Application Obligations

The main obligations of the system would be:

- To allow the client to add/edit/delete files in the repository.
- To process the test request from the user and to trigger the automated build and testing process respectively
- To notify the client if the build fails.
- To generate the logger files for the build process and to store those logger files in repository for further references.
- To notify the client if test execution process fails. Though the test execution process completed successfully, it must notify the client with the status of test cases.
- After execution of test cases, the test logs must be stored in the repository for developers reference to fix the problem

2.3 Organizing Principles

The organizing principles are to perform the primary functionalities of the continuous integration build server through repository, build server and the test harness modules while other functionalities are performed using separate isolated modules which will communicate these three modules.

3 USE CASES

3.1 Actors

3.1.1 Developers

Developers integrate their code into a shared repository several times a day in order to obtain rapid feedbacks. Developers can get the latest and updated code from repository before starting the development work. Though, many developers may work in the same files and so the files can be updated any time before the developer finishes his work.

3.1.2 Managers and Team Leaders

Check status of builds and tests by notifications from the application. And if there are any failures at the time build or test, the captured loggers in the repository helps them to analyze and identify the associated with the failure scenarios. They send the rapid feedback to developers so that the developers can start fixing it as soon as possible.

3.1.3 Quality Assurance and Testing Team

They create the test drivers which consists of all the possible test cases and by automating this testing process, it save lots of man hours. The application replace many QA process. They can find any sort of defect by sending the test request to the application and once they find, they notify developers as fast as possible.

3.1.4 Other Development Tools

Tools such as Continuous Delivery and Continuous Deployment can use the tested defect free build for moving the build to the production environment.

3.1.5 Instructors and Teaching Assistants

Instructors and Teaching Assistants will perform testing tasks on this system. They formally witness that all the requirements of the Build server are fulfilled. They are also responsible for identifying missteps in the design of the system, if there are any, and to notify them to the developers of this system.

3.2 Uses

Build server with continuous integration was introduced to overcome the limitations of the manual integration and testing process. Few of the use cases are:

- Build server with continuous integration benefits the developers most because it allows for code produced to be automatically tested and continuously integrated with developer’s code, and with the existing codebase. The developer benefits from receiving continuous and immediate feedback regarding code and integration errors. As the error fixes, automated testing tools in this stage will report if the errors were successfully fixed and when the code is accepted. This continuous feedback loop dramatically increases a developer’s productivity.
- Integrating the code more frequently leads to reduced risk levels on any project.
- Due to automated testing, most of the errors will be identified during the testing process and so there are less number of chances to get the defects during staging or in production environment.
- Automation process reduces the labor on repetitive processes, making the people to do higher-value work.
- Build server with continuous integration can enable us to release deployable software at any point in time.
- It decreases the code review time as the new files can be merged with the older one using version controller.

5. APPLICATON ACTIVITIES

The Key task of the application is to automate the build and testing process. When many works on the files for the development, it will be difficult for them to integrate and test manually. So this Continuation integration build server helps the team by automating the integration and testing processes.

The overview of the flow of activities for this project is expressed in the following activity diagrams.

The main activities involved in the application are described below

4.1 Getting Filenames from Repository

The client GUI allows user to browse through the repository and to make test requests for the selected files. So the files available in the repository are listed in the GUI. To make this list, a filename request message is sent to the repository for getting the list of files in the repository. The repository will reply with the list of names which can be listed in the GUI. File names of both the source files and the saved test requests can be obtained through this activity.

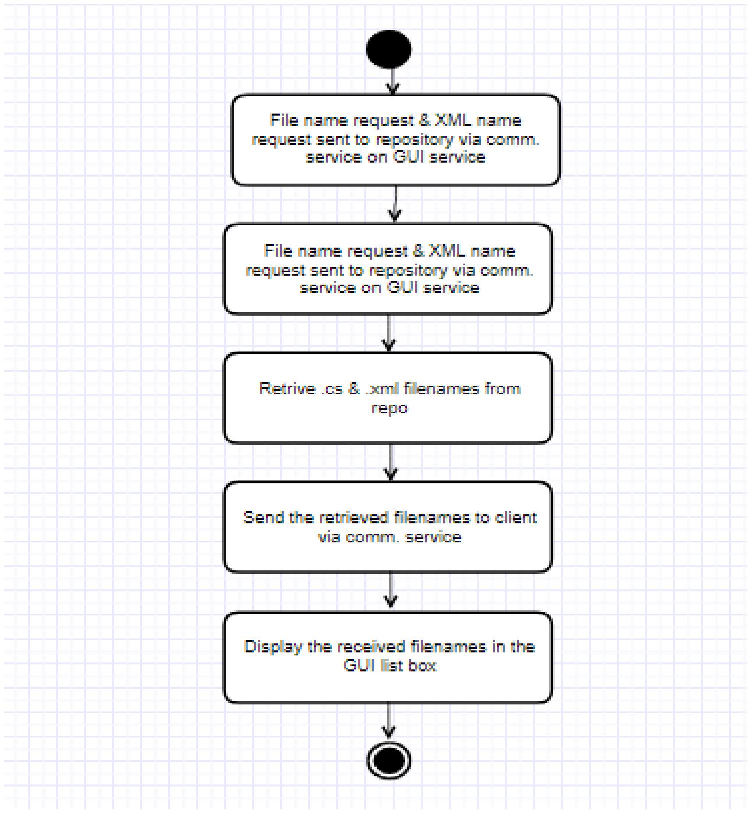

File name request & XML name request sent to repository via comm. service on GUI service
File name request & XML name request sent to repository via comm. service on GUI service
Retrive .cs & .xml filenames from repo
Send the retrieved filenames to client via comm. service
Display the received filenames in the GUI list box

Figure1: Activity Diagram for loading the file names from repository

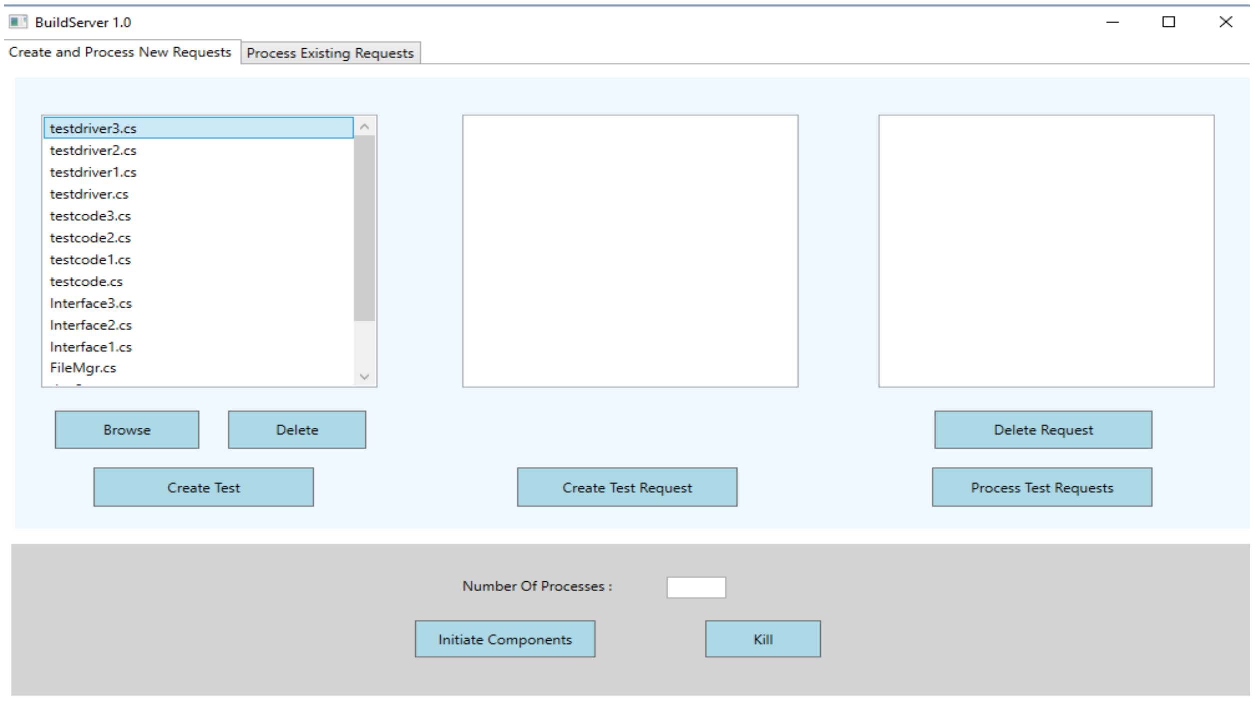
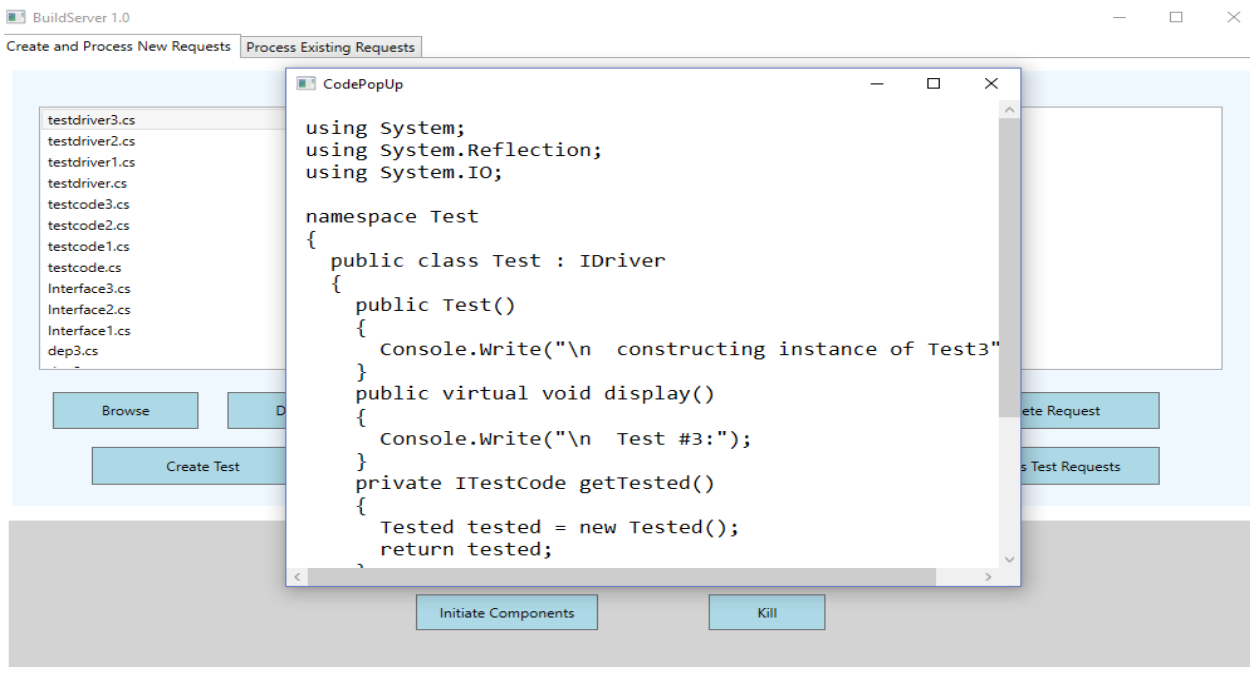


Figure2- Screen on load- received file names on the List Box

4.2 Open and View the repository files

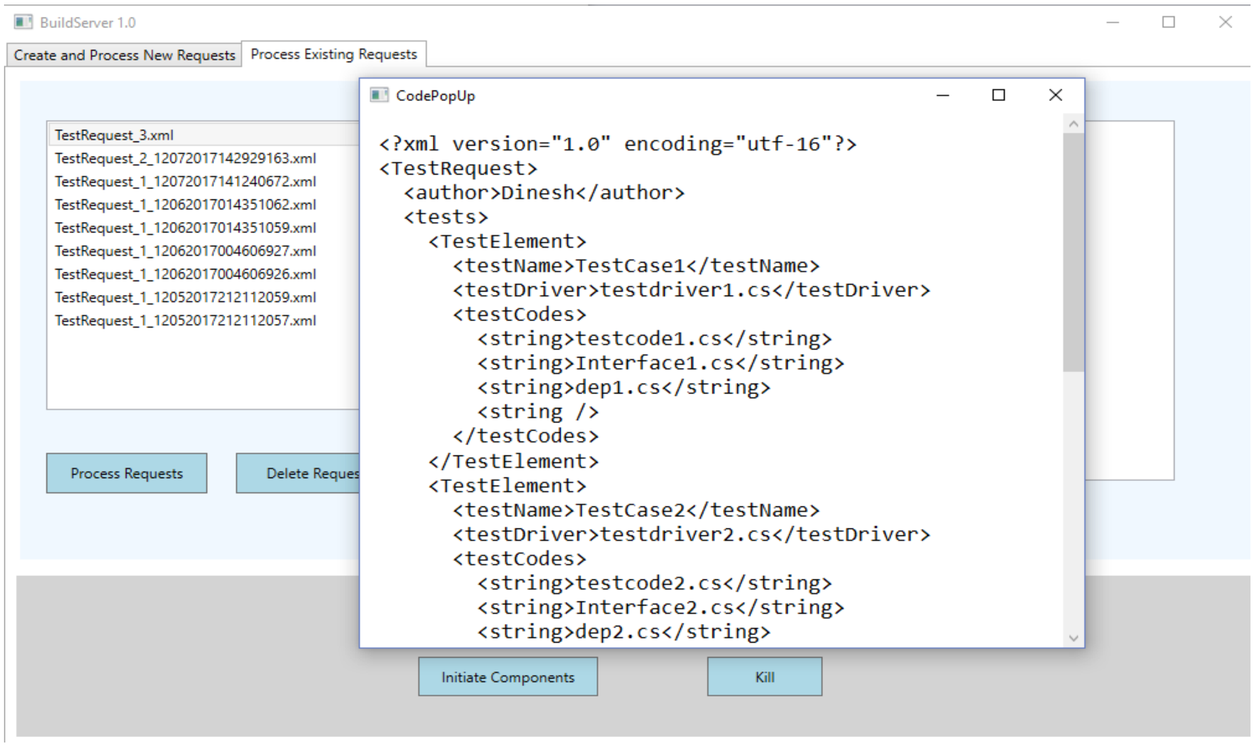
Files in the repository can be opened and viewed through the Client GUI. The client will send the view contents message with selected file name from the GUI to the repository. The repository will read the contents of the received file name and send the whole content as a string to the client. On receiving the contents from the repository, The Client GUI will display the contents in a new Pop-up window. Both the source files and the test requests can be opened and viewed in the GUI.



```
using System;
using System.Reflection;
using System.IO;

namespace Test
{
  public class Test : IDriver
  {
    public Test()
    {
      Console.Write("\n  constructing instance of Test3"
    }
    public virtual void display()
    {
      Console.Write("\n  Test #3:");
    }
    private ITestCode getTested()
    {
      Tested tested = new Tested();
      return tested;
```

Figure3- Opening source files in repository


BuildServer 1.0
Create and Process New Requests
Process Existing Requests
TestRequest_3.xml
TestRequest_2_12072017142929163.xml
TestRequest_1_12072017141240672.xml
TestRequest_1_12062017014351062.xml
TestRequest_1_12062017014351059.xml
TestRequest_1_12062017004606927.xml
TestRequest_1_12062017004606926.xml
TestRequest_1_12052017212112059.xml
TestRequest_1_12052017212112057.xml
Process Requests
Delete Reques
CodePopUp
Initiate Components
Kill

```
<?xml version="1.0" encoding="utf-16"?>
<TestRequest>
  <author>Dinesh</author>
  <tests>
    <TestElement>
      <testName>TestCase1</testName>
      <testDriver>testdriver1.cs</testDriver>
      <testCodes>
        <string>testcode1.cs</string>
        <string>Interface1.cs</string>
        <string>dep1.cs</string>
        <string />
      </testCodes>
    </TestElement>
    <TestElement>
      <testName>TestCase2</testName>
      <testDriver>testdriver2.cs</testDriver>
      <testCodes>
        <string>testcode2.cs</string>
        <string>Interface2.cs</string>
        <string>dep2.cs</string>
```

Figure4- Opening remote test requests

4.3 Deleting the remote files

The files in the repository can be removed from the client GUI. The client can select multiple files from the file list and send a delete request to the repository. On receiving the delete request, the repository will delete the corresponding files in the repository storage and send a notification to the client. The client will update the list of files upon corresponding notification.

4.4 Upload files to repository

The Client GUI allows the user to select and upload the source files to the repository using file request. The communication service helps the client to send the local files to the repository. After sending the last file from the client selection, the client refreshes and updates the file list in the GUI.

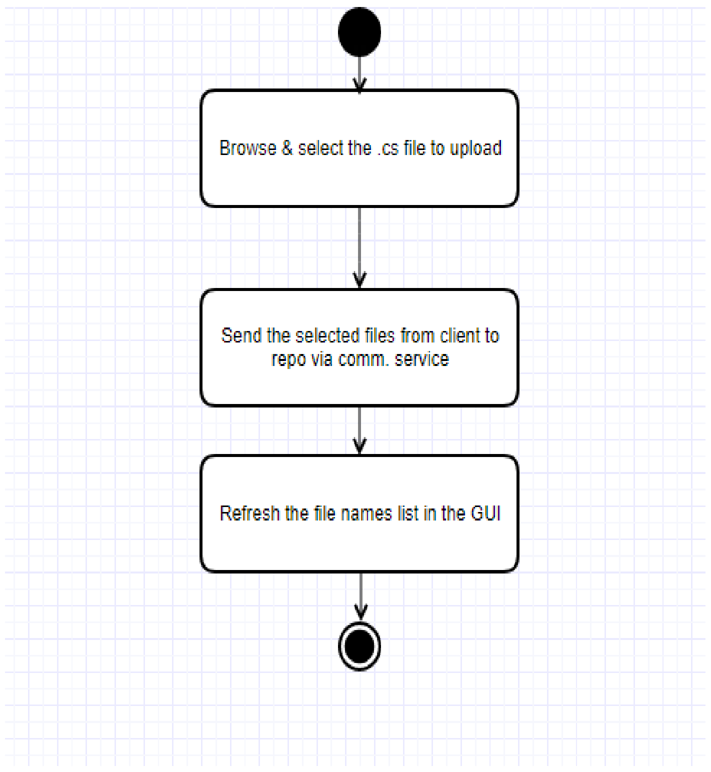
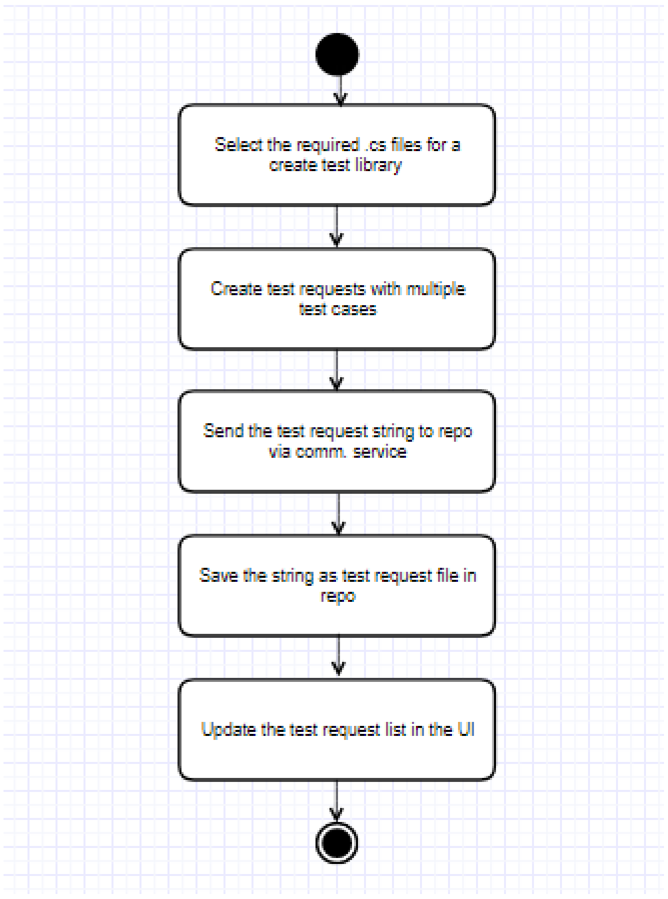

Browse & select the .cs file to upload
Send the selected files from client to repo via comm. service
Refresh the file names list in the GUI

Figure5- Activity diagram for uploading files

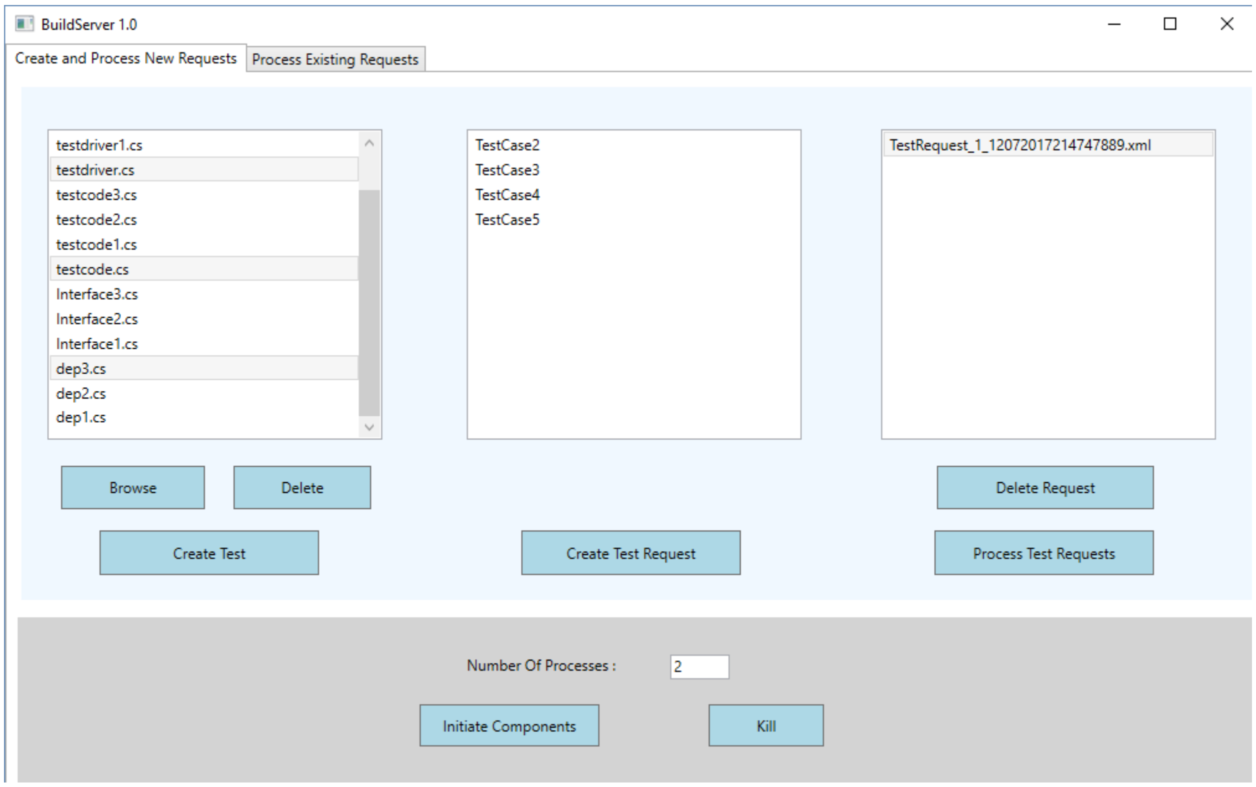
4.5 Creating Tests and Test Requests

The Client GUI allows the user to create test requests for building the test libraries and testing. The user can select the required files for each test cases and create number of tests. And also the user can select multiple test cases and create test requests. All the test request strings will be sent to the repository with create test request command. The repository will create the test request files with the received strings.



Select the required .cs files for a create test library
Create test requests with multiple test cases
Send the test request string to repo via comm. service
Save the string as test request file in repo
Update the test request list in the UI

Figure6- Activity diagram for creating test requests



testdriver1.cs
testdriver.cs
testcode3.cs
testcode2.cs
testcode1.cs
testcode.cs
Interface3.cs
Interface2.cs
Interface1.cs
dep3.cs
dep2.cs
dep1.cs

TestCase2
TestCase3
TestCase4
TestCase5

TestRequest_1_12072017214747889.xml

Number Of Processes : 2

Figure7- Creating test requests in GUI

4.6 Initiate Child process

The client can create required number of child builder processes from the client GUI. The client GUI will send the create child request to the mother builder with the required number. On receiving this request, the Mother builder process will create the number of child processes for building the test libraries. The newly created child processes will send a ready message to the mother builder for building process.

4.7 Allocating work to child process

The Client can select the number of test requests and request the mother builder for processing. On receiving the request, The Mother Builder will request the repository for the test requests and en-queues the received test requests in the test request queue. The Mother Builder will check for the available child process from the ready queue. If any child process is available, the mother builder will de-queue the test request and send the test request to the available child process.

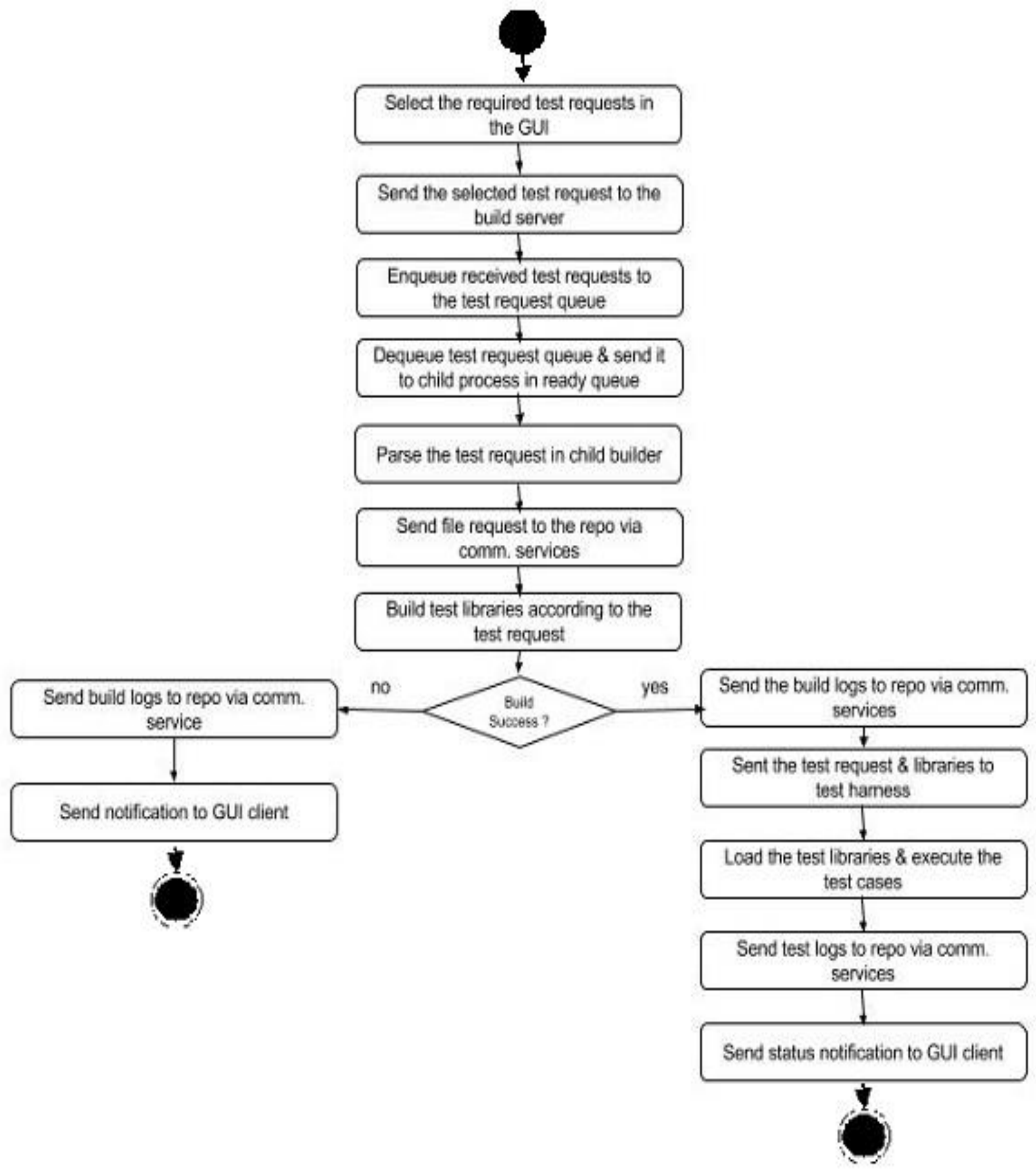

Select the required test requests in the GUI
Send the selected test request to the build server
Enqueue received test requests to the test request queue
Dequeue test request queue & send it to child process in ready queue
Parse the test request in child builder
Send file request to the repo via comm. services
Build test libraries according to the test request
Build Success ?
no
yes
Send build logs to repo via comm. service
Send notification to GUI client
Send the build logs to repo via comm. services
Sent the test request & libraries to test harness
Load the test libraries & execute the test cases
Send test logs to repo via comm. services
Send status notification to GUI client

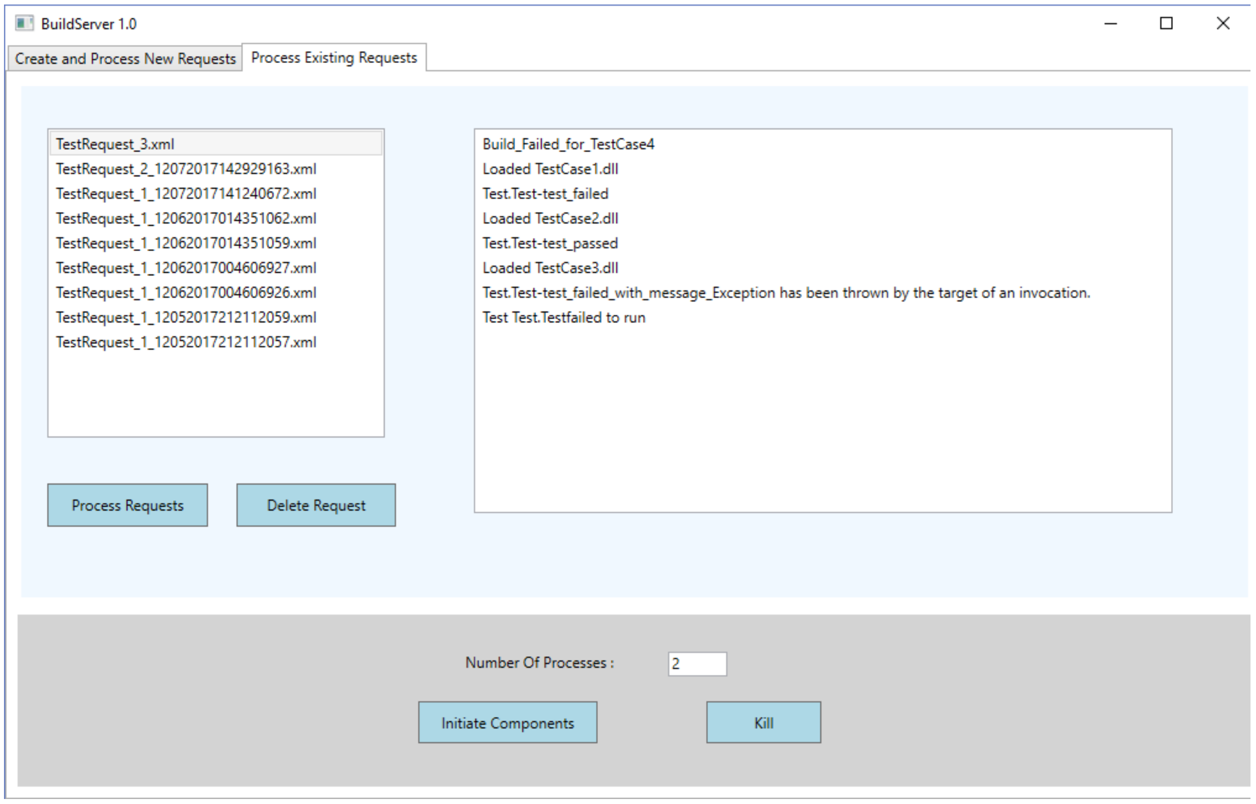
Figure8- Activity diagram for the build process

4.8 Build Process

The child builder accepts the test request from the mother builder. The Xml parser helps the child process to parses the test request. It requests the repository for the files in the test request by sending a file request to the repository. It creates a temp directory where it stores all the received files from the repository. After finishing all these initial process, the child build server attempts to take build with the received files. The loggers inside the build server logs all the activities, process and attempts in the process of making build and at the end of process, it send the log files to the repository through communication service for future reference. If the process succeeds, it will send test request to the test harness for further process. Else, it sends an appropriate notification to the client.

4.9 Testing process

On completion of build process, the child build server sends the test request and the corresponding test libraries to the test harness. The test harness will en-queue the test request in process queue for making the testing process in a sequential manner. Also it stores the received test libraries in a temporary storage for processing. In Parallel, The test harness de-queues the process queue and starts the testing process. It will execute all the test cases one by one. The loggers logs all the process and attempts in the logger files. These logger files are sent to the repository once the testing process ends. Also, the test harness sends the status notifications to the client about each test cases.



```
TestRequest_3.xml
TestRequest_2_12072017142929163.xml
TestRequest_1_12072017141240672.xml
TestRequest_1_12062017014351062.xml
TestRequest_1_12062017014351059.xml
TestRequest_1_12062017004606927.xml
TestRequest_1_12062017004606926.xml
TestRequest_1_12052017212112059.xml
TestRequest_1_12052017212112057.xml
```

```
Build_Failed_for_TestCase4
Loaded TestCase1.dll
Test.Test-test_failed
Loaded TestCase2.dll
Test.Test-test_passed
Loaded TestCase3.dll
Test.Test-test_failed_with_message_Exception has been thrown by the target of an invocation.
Test Test.Testfailed to run
```

Figure9- Receiving Notifications from build server and test harness

4.10 Kill Child Process

The client GUI allows the users to kill the child process at any time. The client will send a Quit message to the mother builder process. On receiving the quit message, the mother builder will send a quit message to all the active child process. When the child process receives the quit message from the mother builder, the child process stops all the building process, close the connection and terminates itself.

5. MESSAGE FLOW ARCHITECTURE

As the System developed in distributed way, Message passing communication between each of the components plays a vital role in utilizing the application. All the components in the system will work without flaw only by receiving the appropriate messages to each of the component at the right time.

The overview of the flow of activities for this project is expressed in the following Message flow diagrams.

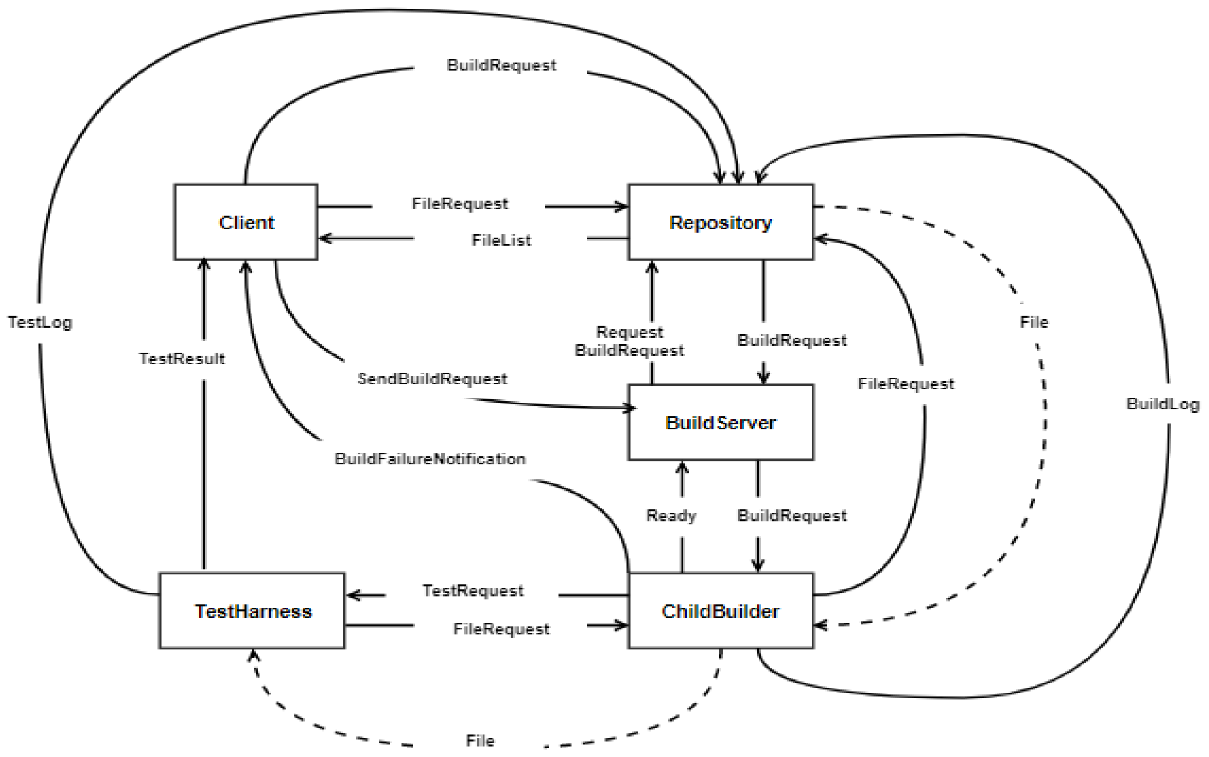


Figure10- Message flow between the components

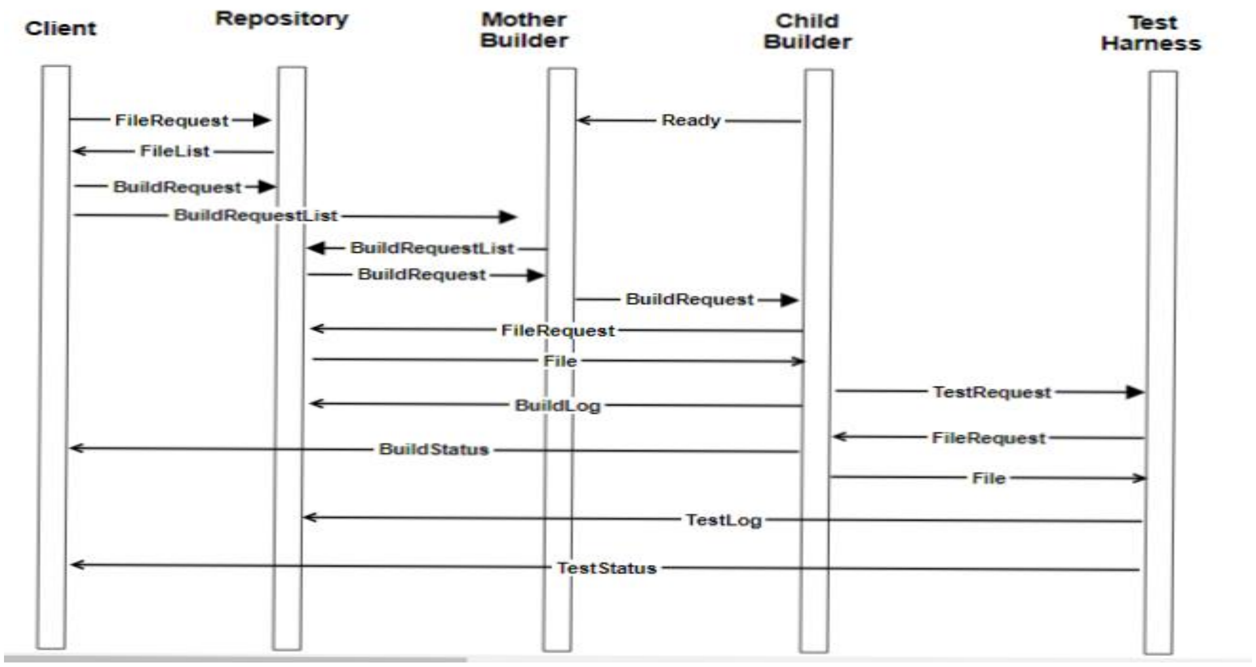


Figure11- Message flow sequence diagram for building process

These are the important messages that are used for building process. The file request from the client is used to fetch the file names from the repository and the repository will send the filenames back in FileList message. The BuildRequestList message from the client to Mother builder is to request the mother builder to start processing the request. The Mother builder sends the BuildRequestList message to repository for getting the build requests from the repository. The repository will send the requested build request files to the mother builder.

The mother builder sends the BuildRequest message to each of the available child builders. The Child builders parse the build request and sends file request to the repository. In reply back to the filerequest message, the repository will send the corresponding requested files to the child builders. On successful completion of building process, the child builders sends the buildlog message to the repository with the build logs. Also, the child builder will send a BuildStatus message to the client which acts as a notification message in GUI.

In Parallel, on finishing the build process, the child builders will send the test request to the test harness. On receiving the test request message, the test harness will en-queue the test request message in the process queue. The test harness will de-queue the process queue and will send a filerequest message to the corresponding child builder. The child builder will send the requested files to the test harness. On completing each test cases, the test harness will send a teststatus message to the client GUI and testlogs message to the repository for storing the logs in repository.

Apart from these, there are some more messages which are used for additional functionalities. The delete file request from the client GUI to the repository is used to request the repository to delete the remote files in the repository. The content view message from the Client GUI to repository is used to open and view the contents of the remote file. The repository will get the file contents and responds back to the GUI with the file content.

6. PARTITIONS

This section describes each package in the application which includes a statement of their responsibilities and their interaction with other packages

Below is the package diagram for Continuous integration Build server

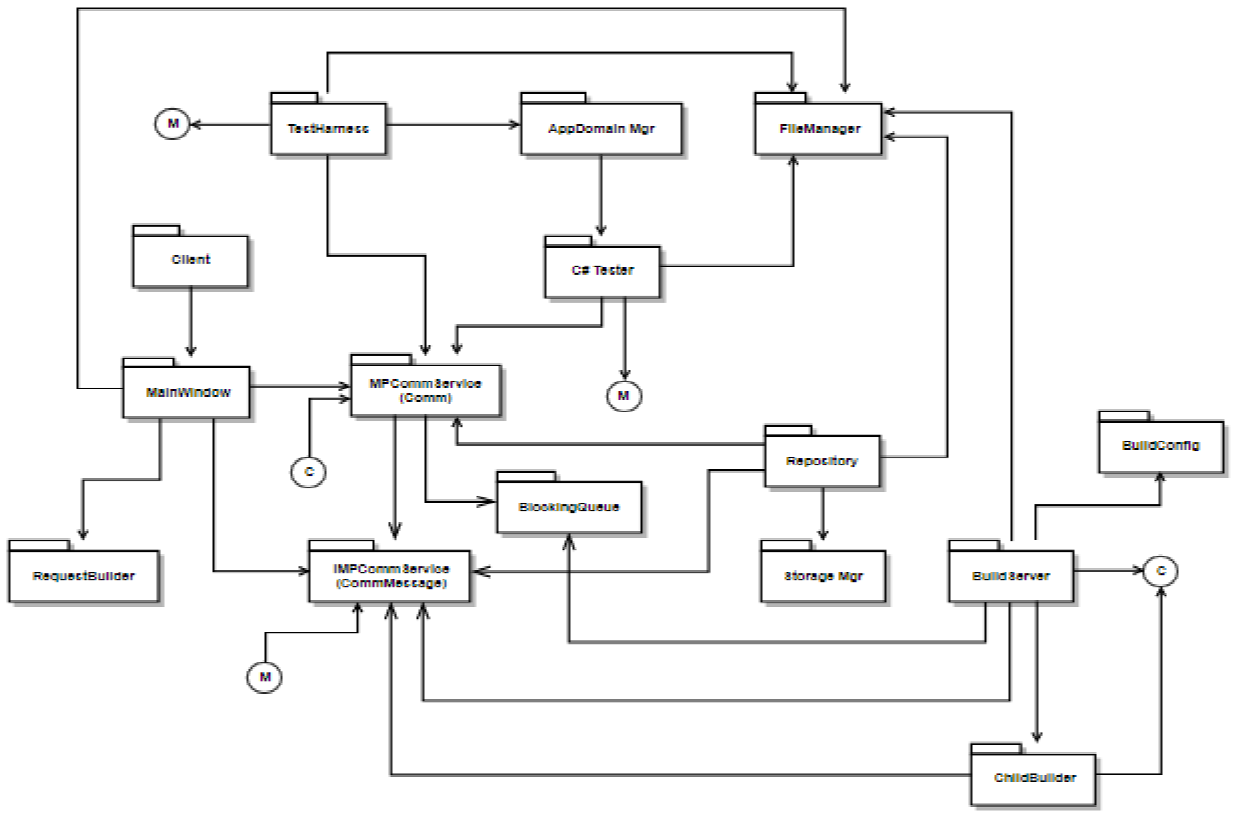


Figure12: Package Diagram for Build Server

6.2 Client GUI

Client GUI is the user interface of the client built with communication service. It make the connection between the repository, builder and test harness which helps the client to make the continual integration and testing automation process. It helps the user to create check in and out the files to repository server. It also allows the user to create test requests for which can be used for building and testing process. It helps the user to open and view the files in the repository.

6.3 MPCommService

MPCommService plays an important role in developing the system in distributed way. It allows all the components in the system to communicate between each of them using communication services. It helps to create the communication channel in all components which makes the system distributed.

6.4 IMPCommService

IMPCommService is the service interface for the message passing communication between the components. It allows the components to create message in the defined format and also it allows the client to configure the paths and addresses of all the components.

6.5 Repository Server

Repository Server is the place where all the files are stored and organized. Client sends all the files and test request to repository. And the repository sends the test request and the files to the Build server. Repository is also used to store the build logs and test logs which are generated during the building and testing process. The client can access those files for fixing those defects.

6.6 File Manager

File manager subsystem play intermediately between the user interface and checkinout manager. While the client selects the files In GUI, the files are pushed to the file manager subsystem. The file manager sends the files to Checkinout manager for further process.

6.7 Build Server

Build server acts as the mother builder for the user entered number of child processes. It creates the number of child processes to make the build process work done. It plays a vital role in continual integration process as it makes gets the responsibility to finish the user requested work. It gets the test request from the user and allocates the work to all the child process and makes the building process fast and more reliable.

.

6.8 Child Builder

Child Builder receives the test request from the Mother builder in Xml format. Parsing the request, the child builder gets the files from the repository and it attempts to build those files. The Child builder creates the build logs and send those logs to repository. If build process fail, it will send a notification to the client. And if the build process succeeds, it will send the test request and built test libraries to the test configurer and commands the test harness to execute the tests.

6.9 Test Harness

Test harness is the vital part of testing and Quality Assurance. Test Harness will get the test request and test libraries from the builder. It configures the Tests by isolating each tests. It loads the libraries from the test Configurer and start the testing process. It executes all the test cases and logs all the testing information using logger. If test execution fails, it notifies the client with appropriate information and stores the test logs to repository. Else, after executing all the test cases, it stores the test logs to repository. After successful completion of each testing, it notifies the client with relevant message.

7. CRITICAL ISSUSES

7.1 Authorization and authentication

One main critical issue related to security is authentication and authorization. Secure authorization and authentication mechanism must be provided for all the users. The privileges must be assigned to each of the user to perform any actions in both the build server and the repository.

7.2 Inconsistent Input Data

Each user can insert the data differently in the xml file of test requests. And the values might be inconsistent or not having the mandatory fields or filled the wrong type of data. An application specific schema for the inputs must be designed and the users must be requested to form the test request in that specific form.

7.3 Version Control

Version control is used to manage large software systems in order to provide them the sense of consistency and organization. This issue can be fixed by creating a module that ensures a versioning of the all the files each time when the code is modified.

7.4 Overload on test harness

Test harness system can be overloaded due to the multiple inputs from the multiple child processes. It can slower down the overall process and decrease the productivity for the users. So, the test harness can also be broken down into multiple processes so that the testing process can be done in parallel.

8. CONCLUSION

In conclusion, the build server with continuous integration can be used for integration and testing purpose. It provides users scalability, flexibility and greater performance on integration and testing process. This OCD explains users and use cases of the system. The package diagram shows the package structure and the interaction between each other to perform all the tasks. It also covers all the activities being carried out by the system using the activity diagram. It describes how the activities takes place in the system and makes easy to understand the system. The message flow diagrams helps us to understand the sequence of message flow and the message dependencies upon the components. We also discussed few critical issue which can result in serious design if not provided by the proper solution. It can be ensured from this OCD that the system can be developed in considerable period of time and with available resources.

9. APPENDIX

9.1 REFERENCES

1. https://ecs.syr.edu/faculty/fawcett/handouts/CSE681/Lectures/StudyGuideOCD.htm
2. https://ecs.syr.edu/faculty/fawcett/handouts/CSE681/Lectures/Project1-F2017.htm
3. https://ecs.syr.edu/faculty/fawcett/handouts/CSE681/code/Project1HelpF2017/
4. https://en.wikipedia.org/wiki/Continuous_integration