Architecture Report

On

Continuous Build and Integration System

CIS 681 : Software Modeling and Analysis

Instructor

Dr. Jim Fawcett

Submitted By

Karankumar Hiteshbhai Patel

(SUID: 874028780)

List Of Content

List Of Figures

1. INTRODUCTION

1.1. Executive Summary

Contiguous Build and Integration System(CBIS) is a robust tool of combining individually tested software modules into the already existing system. In the software industry, there are multiple programmers working independently to build large software system. In such scenario, each developer work on different modules and at the end they integrate it to the existing system to make system productive. For handling this type of task and making software development a smooth process Contiguous Build and Integration System(CBIS) is designed.

CBOS allows multiple clients to work in a collaboration and exchange project related information among each other. It is primarily concerned to ensure that new source code or modifications to an existing source code are successfully tested and checked-in and it is made extractable to all Client after they are successfully checked-in the immutable code repository.

CBIS is a client server architecture that consist of three servers and an arbitrary number of clients. Repository Server stores all of the developing baseline, Test Harness Server runs sequence of tests on module that need to be check-in, Build Server provides build images of the source code that needed by the Test Harness Server for testing it before check-in actually take place. Client has user friendly GUI to access the functionalities of CBIS.

The most obvious users of this tool are Project Manager, Team Leader, Developer, Tester and Software Architect. They use CBIS tool in different way such as for Check-in any module or Check-out any module, to test entire baseline, to view dependency between modules, to view reports of development activities, etc.

At the last, the various critical issues of the CBIS are discussed. They are,

- Communication Failure
- File Contention
- Large Check-in Request
- Huge Check-out Request
- File Sharing
- Load at the Server

1.2. Overall Architecture Diagram

The following figure describes the architectural overview of Contiguous Build And Integration System (CBIS). This architecture is consist of three servers and arbitrary number of clients. Overall idea behind each of this subsystems are explained below.

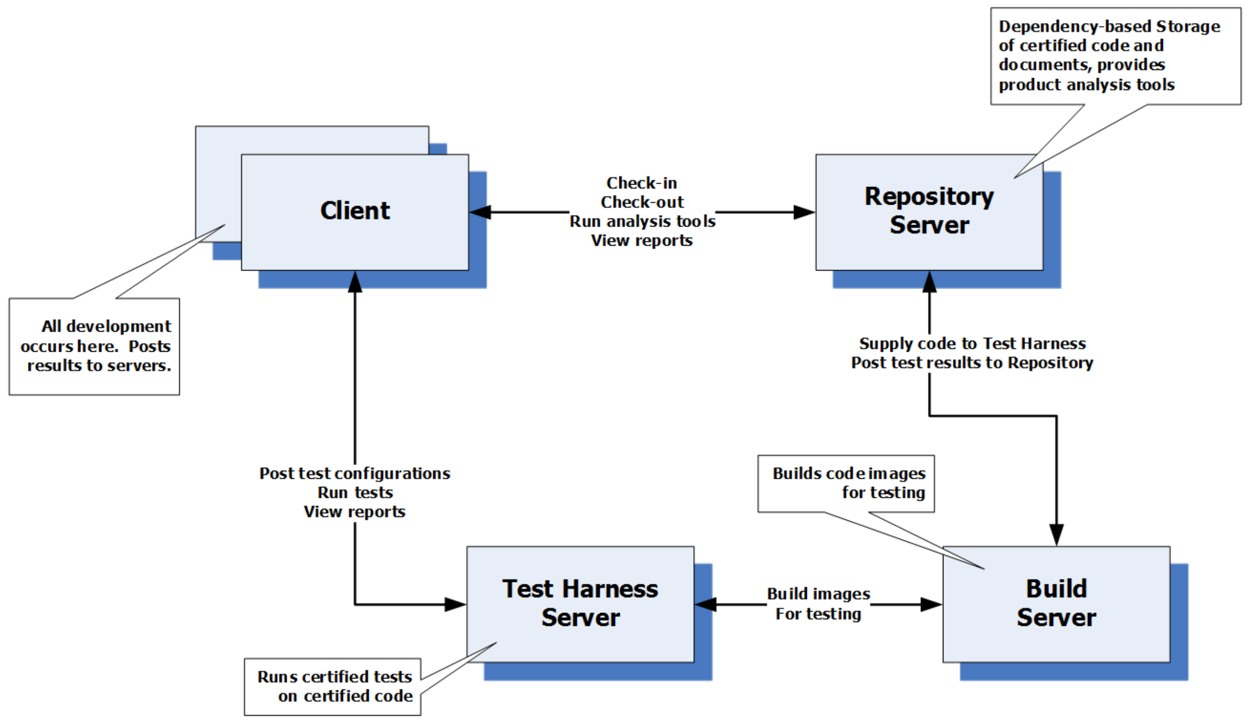

Dependency-based Storage of certified code and documents, provides product analysis tools
Client
Check-in
Check-out
Run analysis tools
View reports
Repository Server
All development occurs here. Posts results to servers.
Supply code to Test Harness
Post test results to Repository
Post test configurations
Run tests
View reports
Builds code images for testing
Test Harness Server
Build images For testing
Build Server
Runs certified tests on certified code

Figure 1 : Architectural Overview Of Contiguous Build And Integration System (CBIS)

The main subsystems of CBIS are as follows:

- *Client*
- *Repository Server*
- *Build Server*
- *Test Harness Server*

Client subsystem is the entry point of our overall system. Client invokes the Repository Server by check-in/check-out request in order to modify or enhance the existing system. Based on the Client request, Repository Server will perform dependency analysis for both of the check-in/check-out requests. It then assigns all dependent code to the Build Server to make a build images for testing. Build Server then pass build images to the Test Harness Server for testing of all dependent code with the test suits provided by the client. Test Harness Server then returns the result log to the Client and Repository Server for both of the possible cases. At the end, Repository Server will add Projects/Packages that are checked-in by the Client to the Baseline in order to enhance or modify the existing system.

Client

Client subsystem is a client application which is installed on different user's machine. The main user's of this application are Developers and Managers. It provides user friendly GUI to the users to access the CBIS. It is mainly use to take input from the user, invoking the servers based on user's request and provides the result back to the user. Client application uses the services of Repository Server, test Harness Server and Build Server.

Main functions of the Client includes :

- Client can download the module from the Repository Server for check-out.
- Client can upload module to the Repository Server for check-in by providing the Test Suit for the module.
- Client receives the result log from the Test-Harness Server & Repository Server.
- Displays enhancement or modification result to the user.

Repository server

Repository Server stores all developing baseline. Every module that tested correctly by the Test Harness Server, will be added to the baseline. If module is not stored in the repository then it is not part of the baseline. Checking-in and Versioning of the modules are the most important tasks of Repository Server which we will discuss later.

Main functions of Repository Server includes :

- Stores modules as a collection of packages, generally it refers as baseline.
- Manages and stores the test suits source code which are needed by the Test Harness Sever.
- Supports check-in, check-out, extraction, and versioning of all modules.
- Finds dependent packages by executing queries about dependency analysis based on the check-in and check-out request.
- It adds orphan packages to the baseline, which are not currently part of the module. It happens only after if test cases provided by the client testified.
- It provides result to the Client regarding new module has been added to the baseline or not.
- Generates XML file that represent the dependent packages of the requested module.

Build server

Build server compiles any code received from the Repository Server, which is required by the Test Harness Server to run the series of tests.

Main functions of Build Server includes :

- Accept build request message from the Test Harness Server.
- Request the Repository Server for the needed source code for each build.
- Maintain Cache to store recently received source code from the Repository Server.
- Builds images as requested and give it back to the Test Harness Server if build is successful.
- Send build error messages to the Repository and Test Harness Servers if one or more builds fail.

TEST HARNESS SERVER

It is responsible for configure and executing test suits provided by the Client. It generates the summary as a result log and sends it to the Client and Repository Server for notifying the client about the check-in status.

Main functions of Test Harness Server includes:

- Frequent testing of new and changed repository source code by executing each test suites on its own thread.
- Load test suits provided by the Client as an XML message.
- Request to Build Server for the compilation of source code that resides into the Repository Server. They should be loaded based on the dependency order maintained by Repository Server.
- Maintain a cache to store previously received compiled module from the Build Server.
- Sends test results, test logs to Repository Server and Client.
- Send selected notifications to registered developers for all modules that depend on the tested modules.

1.3. Organizing Policies for CBIS

The proposed principles and policies of CBIS are discussed below. Some of them are described in details in the following sub sections.

Asynchronous tasks

Our policy suggest that all the internal tasks are asynchronous tasks. User doesn't wait for anything. All the task by the Repository and Test harness server handled asynchronously.

Source code repository maintains all the versions

We must keep our source code repository as immutable so that all the previous versions of that code will remain into the repository in addition to the addition of new code. All these new and old versioning of the code should be maintain whenever any roll-back required. All of these versioning information is maintained in a separate versioning XML file.

Handling concurrent file accesses

The locking mechanism is used for Repository to handle concurrent access on files.
On the user's request of the Checked-out, Repository will lock that files. Hence, allowing only one user at a time to work on current version of a file.

No junk in repository baseline

The Source Code Repository does not allow junk files. Prior to checking-in request by the user inside the source code baseline, the module provided by the user is tested with provided test suits. If these tests fail then it will result in unsuccessful check-in by providing notification to the client.

File caching

To improve the performance, File Caching is the most important factor to be considered. It reduced the amount of the data that communicate between different servers and client. it improves the usability of available code, improve the executing time and overall performance of the system improves. File Caching policy is described in detail in the following 5.4. section.

Check-in policy

Only authorized users called owners are allowed to check-in new modules. Other users will just extract the components. The detailed policy regarding check-in is explained in Repository Server section.

Notifications

Notifications are the indicators that notify the different subsystems of the CBIS regarding the intermediate request or result. User has the functionality to view the notification, mark as read, mark as unread, notify me later, etc.

The type of notifications issued by the different CBIS sub systems are :

- Sending Test Suit pass or fails notification from the Test Harness Server to Repository Server and Client.
- Sending notification after generating the successful build image from Build Server to Test Harness Server.
- Sending notification to different team leaders for building events, meeting requests.
- Sending check-in success or failure notification to the end user from the Repository Server.
- Download Result Log Summary notification to the user from Repository Server.
- Notification regarding new code version release or modification to an existing code version.

2. ACTORS & USES of CBIS

The various actors and the uses of the Contiguous Build and Integration System (CBIS) are described below.

2.1. Users Of CBIS

This section identifies the potential users of the Contiguous Build And Integration System (CBIS). There are various users that can interact with the CBIS in different ways. CBIS is projected to be a perfect tool for following users.

- *Software Developers*
- *Team Leader*
- *Project Managers*
- *Software Architects*
- *Test Teams*

Software developer

In any large software development project, there's strong requirement of understanding the existing code and enhance or modify the present code based on the new requirement. There are hundreds of developers are working towards to achieve this objective of any large software development project. CBIS is very helpful to them to add new packages/modules to the existing project so that new functionalities and new features can be added to the existing project. CBIS is very robust tool providing the access to add new modules to the existing project used to enhance the existing project scope.

Team leader

While dealing with the large software development project, there are number of teams, working toward the achievement of the objective of the software. Team Leaders have been assigned the group of developer to carry out scheduled task. Team Leader can check the performance of each developer of his team by monitoring the success ratio of adding/modifying the new modules by the developer. CBIS can provide such result summary to the Team Leader.

Project manager

All the team leaders work in concurrently and report their progress to the Project Manager. Project Manger is concern about the smooth functioning of the working software in order to meet specified requirement. Through CBIS, Project Manager monitors the total number of successful tests that taken place and can also monitors the proper versioning of the different modules of the project.

Software Architect

When there's a requirement of adding functionality to the existing project, then CBIS allows functionality to the Software Architecture to check-out the existing project. With this option CBIS provides the option of download that project along with the dependent modules that are dependent on that project. So that after getting this, Software Architecture can make an Operational Concept Document for enhancing the existing project.

Test team

Test team makes sure that the project met the entire specified requirement by means of testing with the test drivers. Test Team tests the developer's source code with the test drivers provided by the testers. CBIS is really useful for this purpose, as it provides comprehensive summary of testing any developer's code.

2.2. Uses Of CBIS

There are many applications of Contiguous build And Integration System (CBIS) in which its functionalities can be used. Followings are the potential uses of CBIS:

- *Maintain a Code Repository*
- *Identifying Package Dependency*
- *Software Product Enhancement*
- *Software Product Maintenance*
- *Automate the build/No Human Intervention*

MAINTAIN A CODE REPOSITORY

CBIS helps in maintaining the code repository at Repository Server. It is advisable to integrate the changes to the existing project rather that for multiple versions of the software to be maintain. So the all the integrated modules of the projects are resides at the baseline.

IDENTIFYING PACKAGE DEPENDENCY

CBIS is able to browsing relationship & dependency between the packages of the project, which gives an idea to developer that how these packages are interconnected with each other. This could help a new developer to understand with the new project on which he's going to work on.

Software product enhancement

The most obvious purpose of CBIS is to help the developer to get better understanding of existing code by means of check-out. By going through the CBIS, developer can understand dependency between the different packages of the project. By knowing this developer becomes comfortable for further software enhancement by providing new functionalities and better efficiency. So, CBIS proves the most obvious tool for Software Product Enhancement.

Software product maintenance

Software Product Maintenance is a very challenging task throughout the whole SDLC. In such case we can take CBIS into the confidence to make sure that each maintenance activity not altering any structure of code.

Automate the build/no human intervention

CBIS is an automated tool helps in contiguous integration of new modules to the existing project. Developer just needs to wait until the tests run successfully. All of the integration to the baseline is taken care by the CBIS.

3. CLIENT STRUCTURE

Based on the tasks and the activities at client's end, the Client subsystem is divided into following modules as described below.

3.1. Client Overview

The Client(s) is the subsystem that invokes all the other subcomponents of the CBIS. Client will have GUI through which they can interact with the servers. GUI initiates the actual processing of the CBIS. There are different kinds of Client(s).

- Project Manager
- Team Leader
- Developer
- Testing Team
- Software Architect

Any of the above mentioned Client(s) can may send check-in and check-out request to the Repository Server. The Project Manager and Team Leader may get special functionality based on their roles of viewing build reports of any subsystem reside in Source Code Repository. The main use of Client is to take input from the users and make a client-server request to the server by using WCF communication.

Main features of Client include:

- Different Client will get accessed to pre-defined functionalities via Login to CBIS based on the User Type.
- Client's GUI features with the following panes, which will appear to the User based on their type. They are Check-In, Check-Out, Dependency Analysis, View Reports and Test Baseline.
- Client can make the Check-in request to the Repository Server by providing the new modules with the Test Suits. The Clients get result summary in the Message Log after requesting request to the Server.
- Client can make the Check-out request to the Repository Server by requesting the required modules from the Source Code Repository. Client also has the option of check-outing the dependent files of the requested module.
- Client can see the Dependency Analysis of particular module by selecting it only.
- Project Manager/Team Leader can see the Build report of any module.
- Client can check the entire Baseline by Test Baseline option to carry out schedule maintenance activity.

Different Roles of the Client are discussed below:

Project Manager/Team Leader

- Monitors and Supervise the development activities going on simultaneously
- They may check in any module into the Repository Server via providing test Suits along with the modules.
- They may check out any module from the Source Code Repository by requesting the Module of the Subsystem.
- They may get Dependency Analysis of any module.
- They can generate Build Reports of any Subsystem.
- They may test the entire Baseline to ensure the entire application works perfectly.
- Assign ownership to the User's regarding Check-In module into the Baseline.

Software Developer

- Mainly concern about the development and integration of that module into the source code repository.
- They may check in any module into the Repository Server via providing test Suits along with the modules.
- They may check out any module from the Source Code Repository by requesting the Module of the Subsystem.
- They may get Dependency Analysis of any module.

Testing Team

- Mainly concern about the testing of the subsystems
- They may check out any module from the Source Code Repository by requesting the Module of the Subsystem.
- They may get Dependency Analysis of any module.
- They may test the entire Baseline to ensure the entire application works perfectly.

Software Architect

- Mainly concern about the possibility of how to enhance the subsystem by identifying targeted module's dependencies to other modules.
- They may get Dependency Analysis of any module.
- They may check out any module from the Source Code Repository by requesting the Module of the Subsystem.

3.2. Client Context Diagram

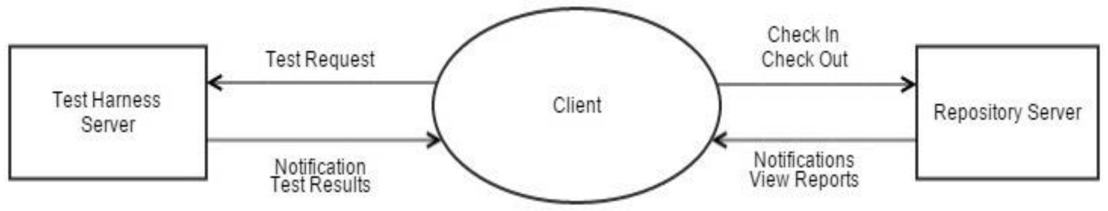


Figure 2 : Context Diagram of Client Subsystem

Context diagram represents the interaction of the Client subsystem with the other subsystems. Repository Server and Test Harness Server are the two subsystem to which Client interacts. Client can be invoked by the any permissible user. User can make request to Client by using Graphical User Interface of the client. Through GUI, Client can request one of the following requests: 1) Check-In and 2) Check-Out.

While Check-in, Client has to provide new modules that he wants to integrate along with the corresponding Test Suits. Client sends this request to both of the server, Test Harness Server and Repository Server.

Repository Server will provide the Source Code of new module through the Build Server to the Test Harness Server. Test Harness Server already have the Test Suits as sent by the Client, he starts testing each files of that module. Based on the result of the project, Test Harness Server & Repository Server notifies the Client via message log and notification that the new module has been added or not. While check-outing, Clients request is made to the Repository Server, and Repository Server will give output based on the user's check-out request. So context diagram represents the interaction of the Client with the Repository and test Harness Server.

3.3 Client Package Diagram

Client Package Diagram shows the Modular Structure of Client Subsystem.

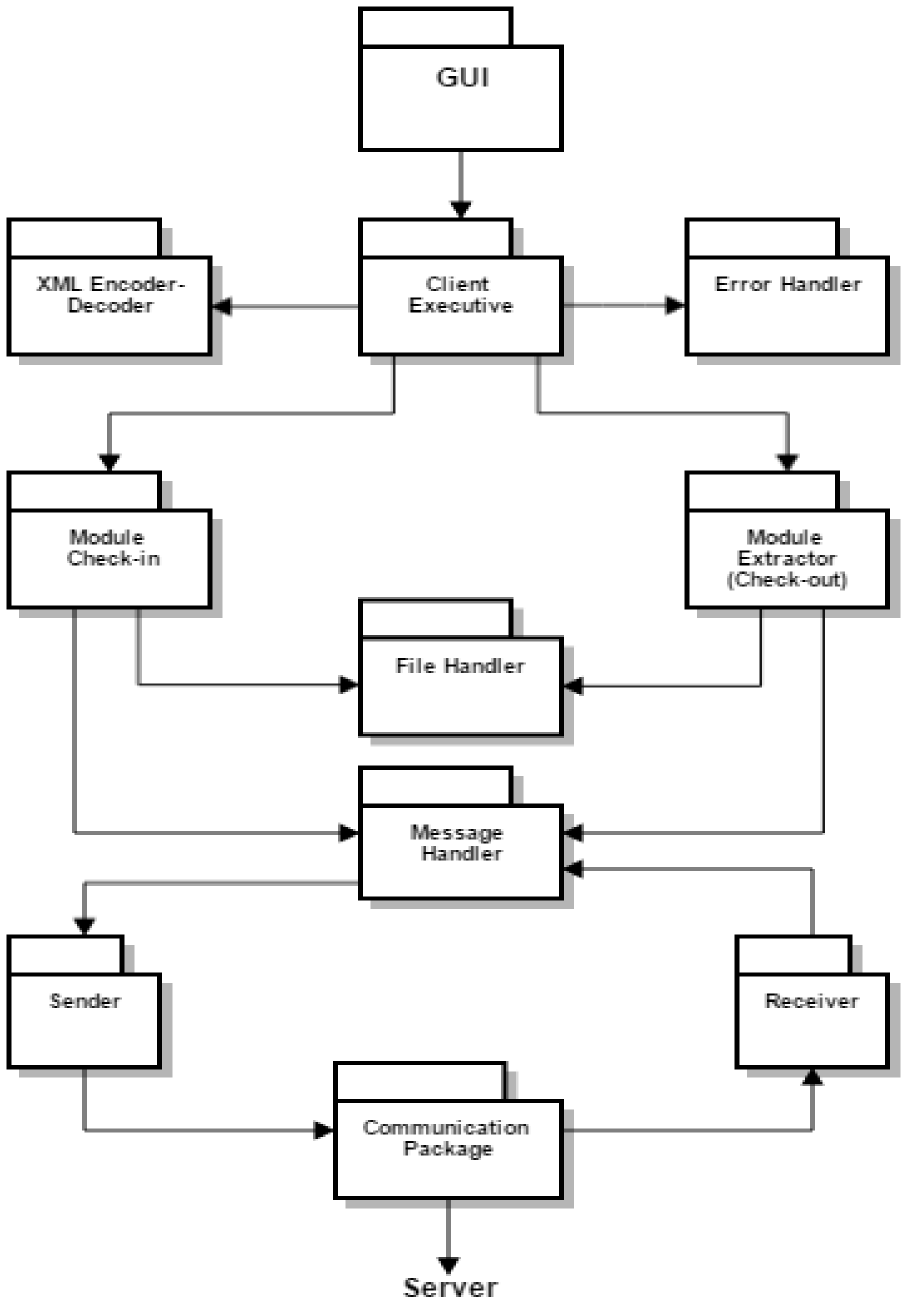


Figure 3 : Package Diagram of Client Subsystem

GUI Package

GUI package provides the interface to the user through which user can interact with the tool. GUI Package interacts with the Client Executive package to pass the user's selected request. It also display the result that is received by the Client Executive in response to the corresponding request.

Client Executive Package

Client Executive Package is the main package of the Client Subsystem. It interacts with all other packages of this system. It passes the request of the user to the appropriate modules for further advancement. And receives the final response from the respective servers.

XML Encoder-Decoder Package

Encoding and Decoding of the XML files are done by this package. This package is useful to map the module names with the corresponding Test Suits. These are mapped here in the XML file by using XML Decoder functionality and send it to the Client Executive back so that it can send this mapped XML file for the further processing. Any Response received from the Server in form of XML file, XML Encoder is used to encode it and passes to the GUI through Client Executive.

Error Handler Package

Error Handler package is responsible for handling exceptions that are thrown by other packages of client module. Once the exception caught, it displays the error message which is easily interpreted by the user. Error Handler package enhance error handling capability of Client Sub System.

Module Check-In Package

Module Check-In package is responsible for passing the user's Check-In request to the server. It receives the user's check-in request from the Client Executive package and generates the request message to send at the Server side throught Message Handler and Communication Package.

Module Check-Out Package

Module Check-Out package is resposible for sending check-out request of the user to server via Message Handler and Communication Package. It receives user's request through Client Executinve and passes the result of this request back to the Client Executive package to display it on GUI.

File Handler Package

File Handle package will retrive the files from and to the system. It invokes by the Check-in and Check-out packages to retrive the files located at the system. It is helpful for finding Test Suits and Corresponding files on the system.

Sender Package

Sender package is used by Client package whenever client wants to make any request to the server. There is a queue maintain by the sender package in which multiple client request pipelined. Communication package takes care of fetching client request from this queue and sends that request to the server for analysis.

Receiver Package

Receiver package is used by Communication package to get back response from the server. Receiver is also maintains queue for storing multiple responses from the server. Client package takes care of fetching server response one by one and do further processing based on user’s request.

Communication Package

It establishes communication channel between the Client-Server Repository, Client-Test Harness and Client-Project Server. Windows Communication Foundation (WCF) is being use here to make communication between Client & Server. It generally provides channel to link client to server. All requests from clients & responses from servers are going through this package.

3.4. Client Activity Diagrams

Following are different activities performed at client side.

1) Login Activity

In order to access the functionalities of CBIS user has to logged-in into the system by providing credentials. The flow of this activity is discussed below.

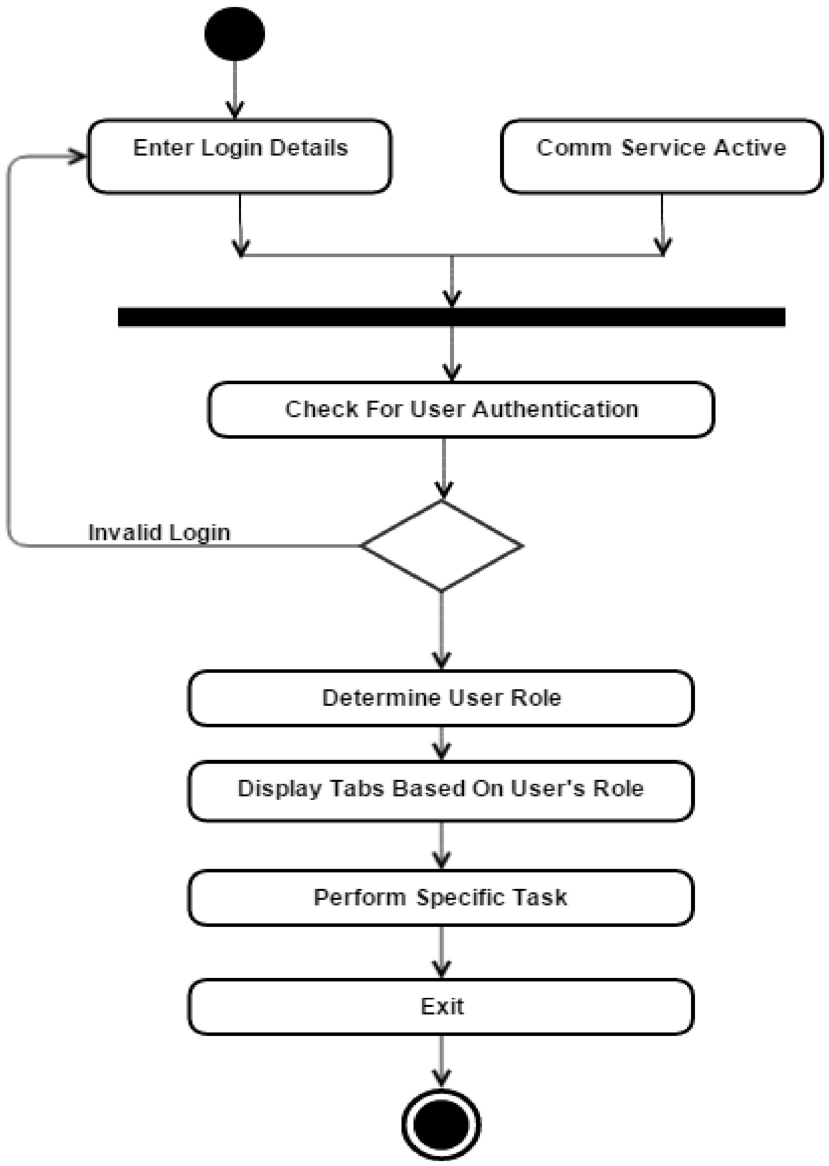


Figure 4 : Login Activity of Client Subsystem

1) Enter login details

User will enter the Login Details by providing User Type, User Name and Password. So that user can access the functionality of the CBIS.

2) Comm Service Active

Once the user clicks on the login activity, client machine will communicate the user request to server via Proxy Channel using communication service.

3) Verify User Authentication

Login information provided by the users will be forwarded to the server for an authentication. Entered credential will be checked at server side. If credential will be verified then user can access the services of the CBIS.

4) Determine User's Role & Assign Functionalities/Tabs Accordingly

The functionalities of the CBIS is varies for users and users. At the time of the login the User's role will be decide and according to the type of the User, different tabs will be open. For example, Project Manager will have access to all of the functionalities of CBIS while Software Architecture only have access to the check-out and carry out dependency analysis.

5) Perform Different Activities/Tasks

After log-in user can perform number of functionalities(Explained in following section). Such as,

- Check-in Module
- Check-out Module
- Carry Out Dependency Analysis
- View Report
- Test Entire Application
-

6) Exit

User exits the application after performing required tasks.

2) Client's Activity With the Server

After successful login, Client can perform number of activities base on their type. The number of different activities performed by Client is explained below.

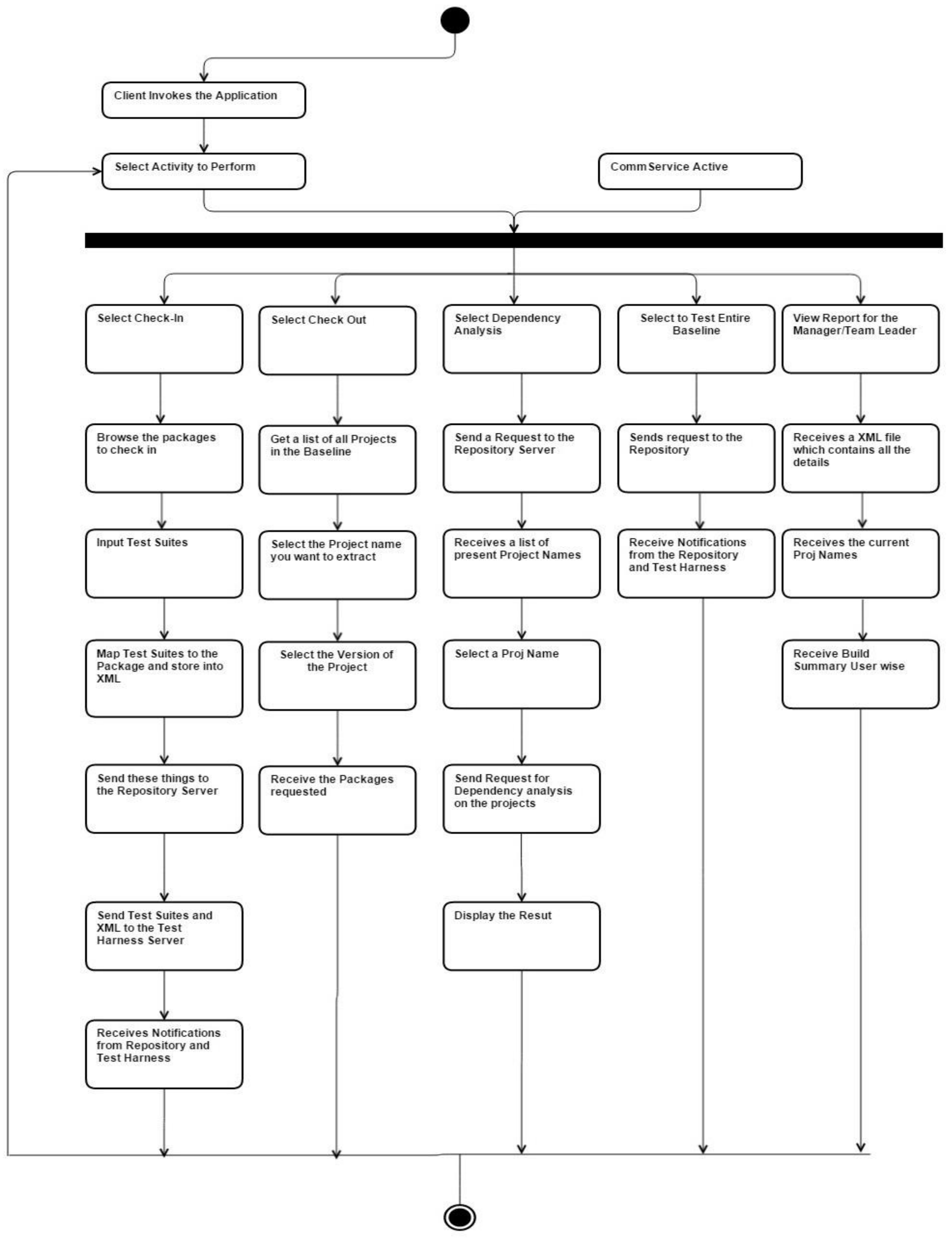

Client Invokes the Application
Select Activity to Perform
Comm Service Active
Select Check-In
Select Check Out
Select Dependency Analysis
Select to Test Entire Baseline
View Report for the Manager/Team Leader
Browse the packages to check in
Get a list of all Projects in the Baseline
Send a Request to the Repository Server
Sends request to the Repository
Receives a XML file which contains all the details
Input Test Suites
Select the Project name you want to extract
Receives a list of present Project Names
Receive Notifications from the Repository and Test Harness
Receives the current Proj Names
Map Test Suites to the Package and store into XML
Select the Version of the Project
Select a Proj Name
Receive Build Summary User wise
Send these things to the Repository Server
Receive the Packages requested
Send Request for Dependency analysis on the projects
Send Test Suites and XML to the Test Harness Server
Display the Resut
Receives Notifications from Repository and Test Harness

Figure 5 : Client's Activity with the Server

1) Client Invokes the Application

User will login to CBIS system by providing credentials.

2) Client Selects Activity/Task

Based on the user type, user will select one of the following CBIS activities. All of these activities are interact with the server.

- Check-in Module
- Check-out Module
- Carry Out Dependency Analysis
- View Report
- Test Entire Application

3) CommService Active

After selecting one of the activities from the GUI, user machine will start communicating with the server via Proxy Channel using the communication service. If for some reasons, server fails then our CBIS application will terminate and will notify the user about failed connection.

4) Select Check-in Module Activity

User can check-in any module to the Source Code Repository. Test Harness Server will execute series of Test Cases for each of the files of module and notifies the result at the end of the testing. The workflow of this activity is as follows:

- User selects any module that he wants to upload to the baseline.
- Along with the module user has to select test suits for the testing purpose.
- The mapping has been done at the client side. Client will map each file with the corresponding test cases and Generates new XML Mapping file.
- Client will then send modules; test suits; and XML Mapping file to the Repository Server and test suits and XML Mapping file to the Test Harness Server for further processing.
- Test Harness Server will execute series of tests on the module, file by file and at the end of the execution of test cases, it will notify user and repository server regarding its result of testing. If tests succeeded, then Repository Server will add check-in new module and sends user the notification message regarding the same.

4) Select Check-out Module Activity

User can check-out any module from the Source Code Repository. In order to do so, following events will happen:

- User will select any subsystem from where he wants to check-out any module.
- After selecting subsystem, user will get list of projects resides on that subsystem.
- Out of the list of the project, user will select particular project and along with its version. User may check-out dependent files of those projects also.
- After selecting it, it waits for the response from the Repository Server.
- At the end, user receives the required package along with the dependent files from the Repository Server.

5) Carry Out Dependency Analysis Activity

User also can perform the Dependency Analysis on particular package. Following events will take place:

- User selects the project names from the Source Code Repository.
- User sends those names to the Repository Server.
- Repository Server will perform dependency analysis on those projects and sends back the result to the user.
- User can able to see result by downloading XML dependency file received from the Repository Server.

6) View Report Activity

Program Manager/Team Leader may use this activity to know the performance of any developer. Build Reports provides the performance analysis of developer.

- User selects subsystem.
- User will get list of the developers working on that subsystem.
- After selecting specific developer, he request at the repository server regarding generating build report which will include the no. of test succeedd or failed, and previous 1-week enhancement or modification history of choosen developer.
- User receives result into the XML file from the repository Server.

7) Test Baseline Activity

User may want to perform any schedule maintenance task through this activity. The number of different steps in this activity is as follows:

- Any authorized user such as Tester, Program Manager and Team Leader will get list of subsystem resides at the Repository Server.
- User will select any entire subsystem, he wants to test.
- Based on the user's request, Test Harness Server will perform the entire testing by using Test Suits on the subsystem by getting source code & XML mapping file from the Repository Server.
- Test Harness Server will notify the Repository Server and User regarding the result of testing.
- User receives the testing report from the Repository Server.

3.5. Client Views

Client's Login Page is the entry point of our CBIS system. The Graphical User Interface(GUI) provides the Login page to access the functionalities of CBIS.

1) Login View of Client

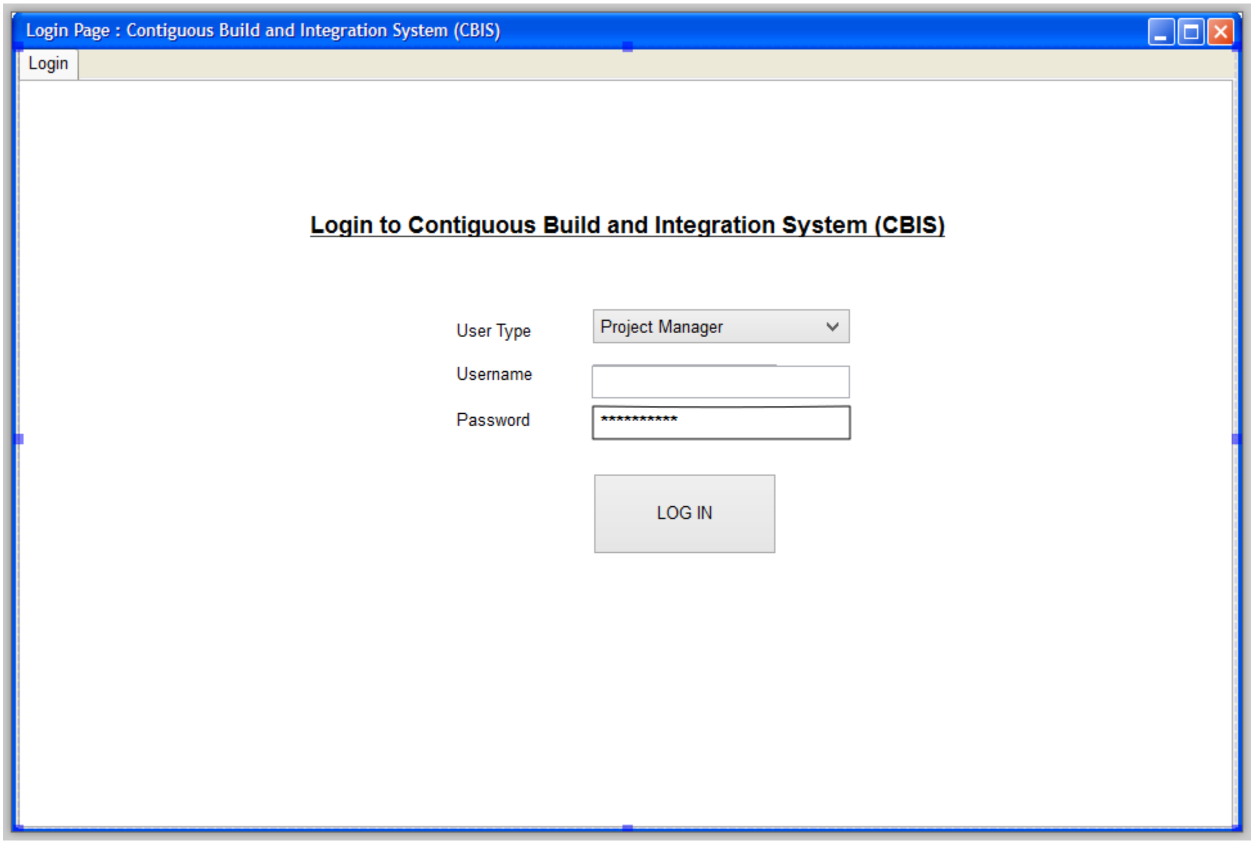

Login Page : Contiguous Build and Integration System (CBIS)
Login
Login to Contiguous Build and Integration System (CBIS)
User Type
Project Manager
Username
Password
****
LOG IN

Figure 6 : Login View of Client

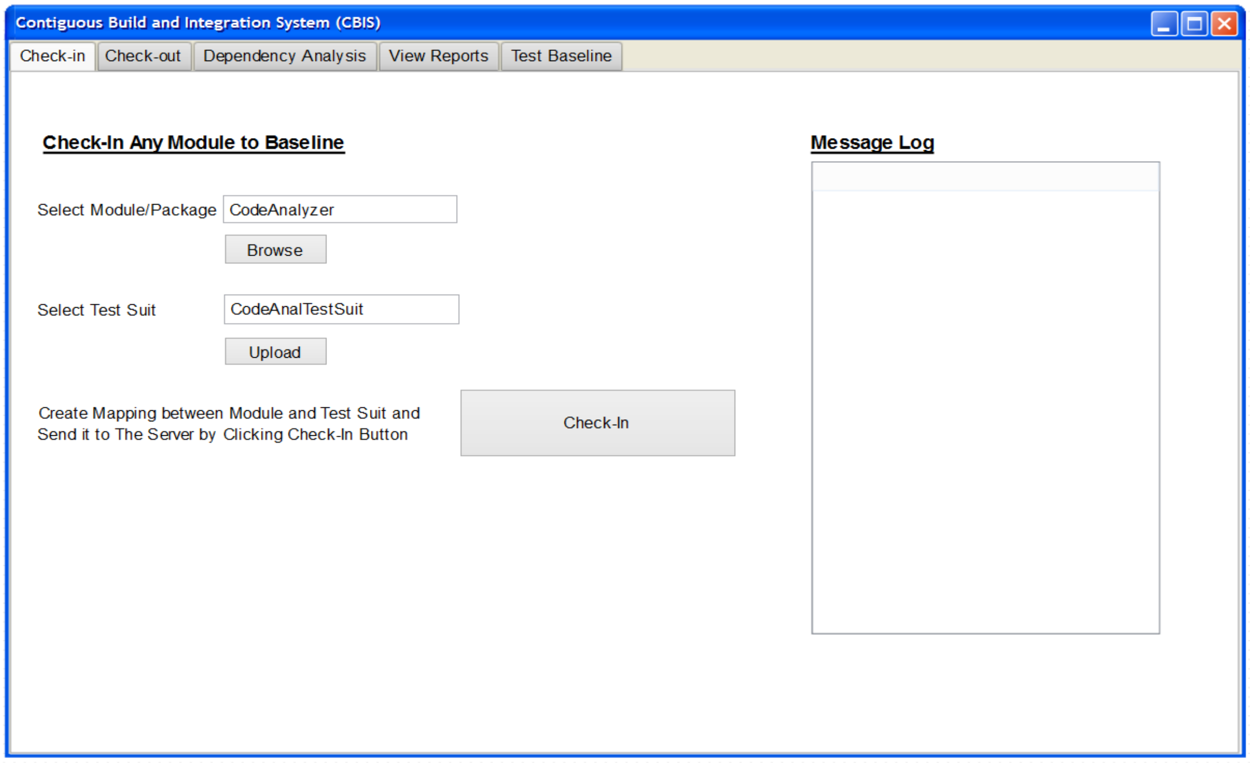
Through this view, user can log in to the CBIS. User has to provide the credentials such as Username and Password along with the User Type. There are number of different types of user, who can log in into the CBIS. They are,

- Project Manager
- Team Leader
- Software Developer
- Tester
- Software Architect

2) Check-In View of Client

This is the view through which any user can Check-In any module to the source code repository. This view is restricted to only some of authorized users. This view can be access by following users.

- Project Manager
- Team Leader
- Software Developer



Select Module/Package: CodeAnalyzer

Select Test Suit: CodeAnalTestSuit

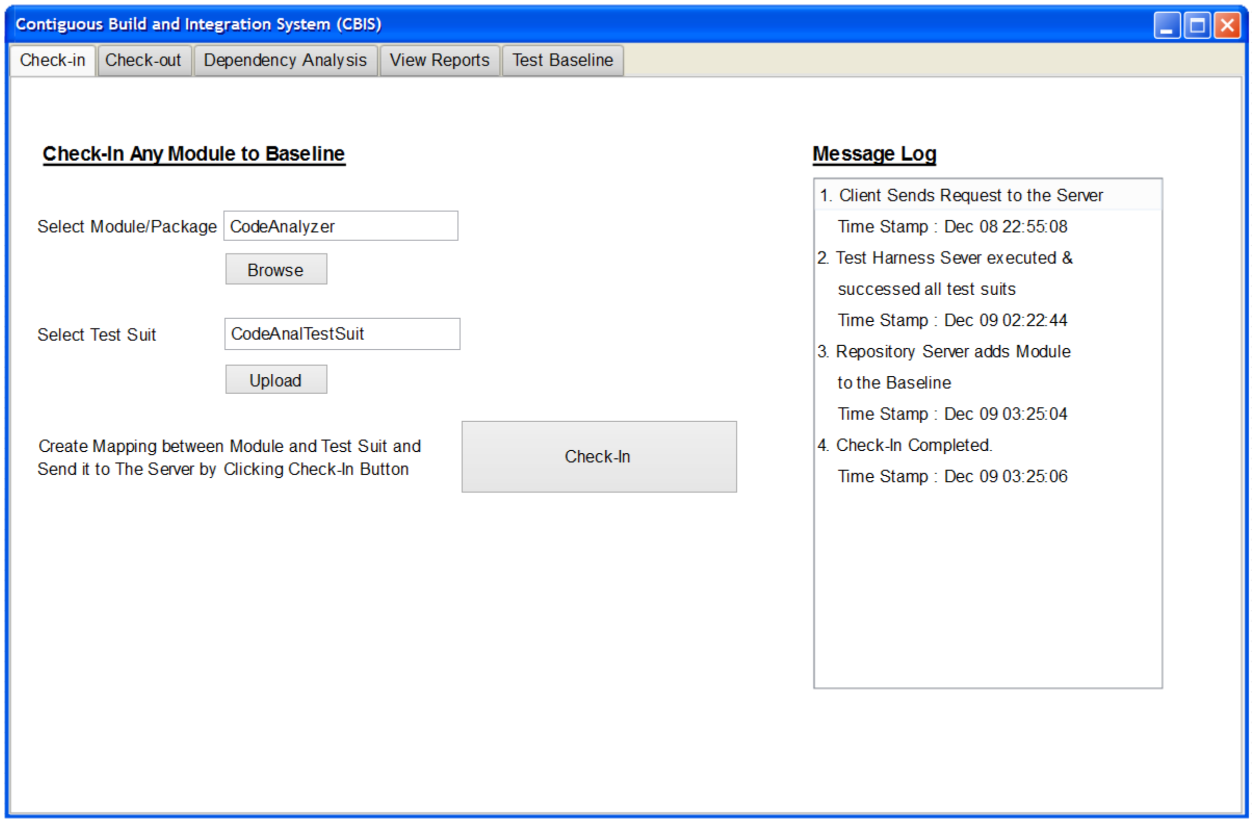
Create Mapping between Module and Test Suit and Send it to The Server by Clicking Check-In Button

Figure 7 : Check-In View of Client

Any authorized user can check-in any module to the source code repository by selecting specific module/package and respective Test Suits.
After selecting Modules and uploading Test Suits, client makes mapping file to map these two files and by clicking Check-in button, Client sends these files, test suits and mapping files to the Servers.

3) Check-In View of Client with Message Logs

After Check-in request is made by the Client, client waits till result receives. In order to do so, Message Log has been maintained to notify Client about which background activity has been completed.



Select Module/Package: CodeAnalyzer

Select Test Suit: CodeAnalTestSuit

Create Mapping between Module and Test Suit and Send it to The Server by Clicking Check-In Button

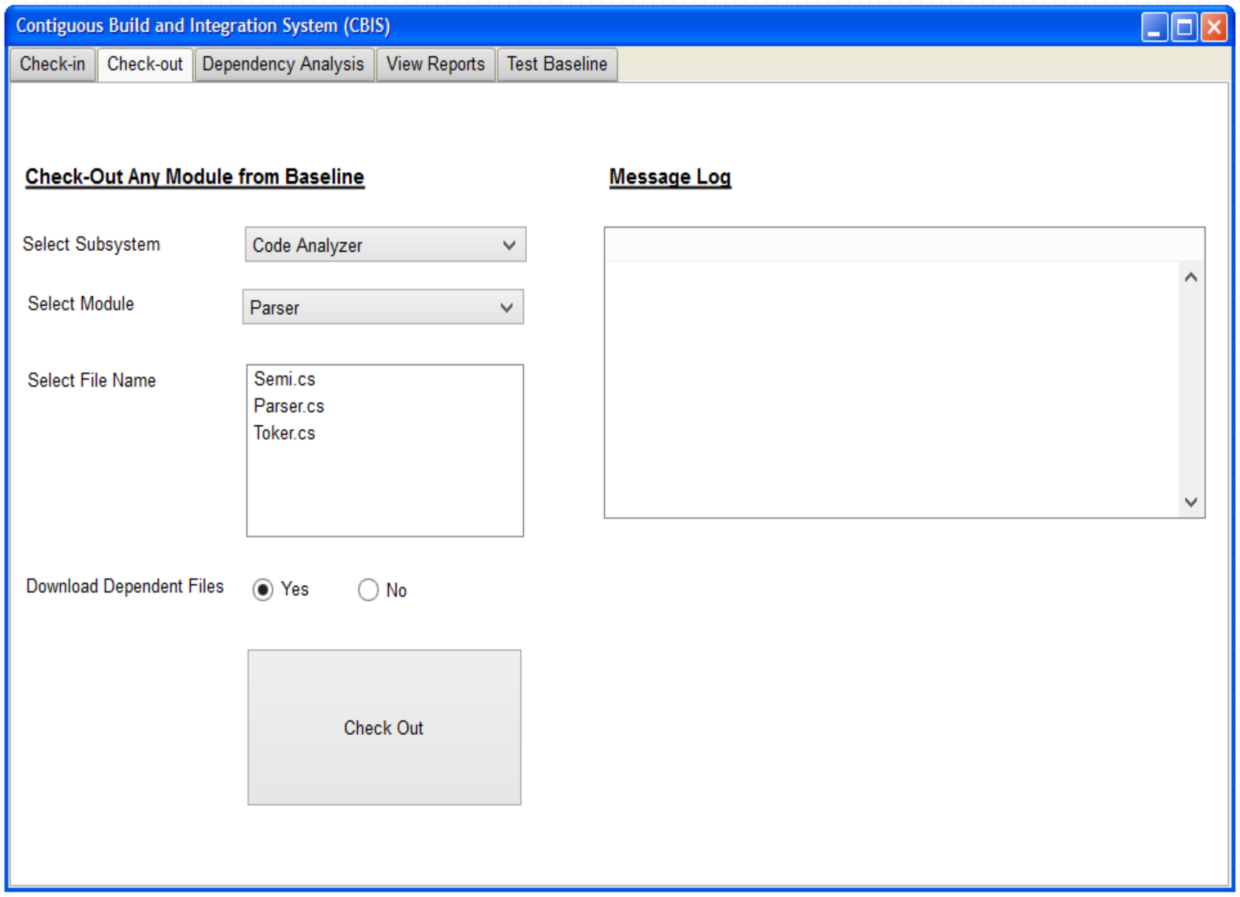
1. Client Sends Request to the Server
   Time Stamp : Dec 08 22:55:08
2. Test Harness Sever executed & successed all test suits
   Time Stamp : Dec 09 02:22:44
3. Repository Server adds Module to the Baseline
   Time Stamp : Dec 09 03:25:04
4. Check-In Completed.
   Time Stamp : Dec 09 03:25:06

Figure 8 : Check-In View of Client with Message Log

Time Stamps are used to let Client know about the timings of the activities completed at the server side. Thus, Message Log is very helpful for the client.
The possible message log that may displayed during Client-Server Communications are:
1) Client Sends Request.
2) Client Failed to make Request to Server.
3) Test Harness Server Executed & Passes all Test Suits.
4) Test Harness Server Executed & Failed to pass following Test Suits
5) Repository Server adds Module into the Baseline.
6) Repository Server failed to add Module into the Baseline.
7) Check-In Completed.
8) Check-In Failed.

4) Check-In View of Client

This is the view through which any user can Check-Out any module from the Source Code Repository. This view is accessible by all type of the user.



Select Subsystem: Code Analyzer
Select Module: Parser
Select File Name: Semi.cs, Parser.cs, Toker.cs
Download Dependent Files: [✓ Yes] No

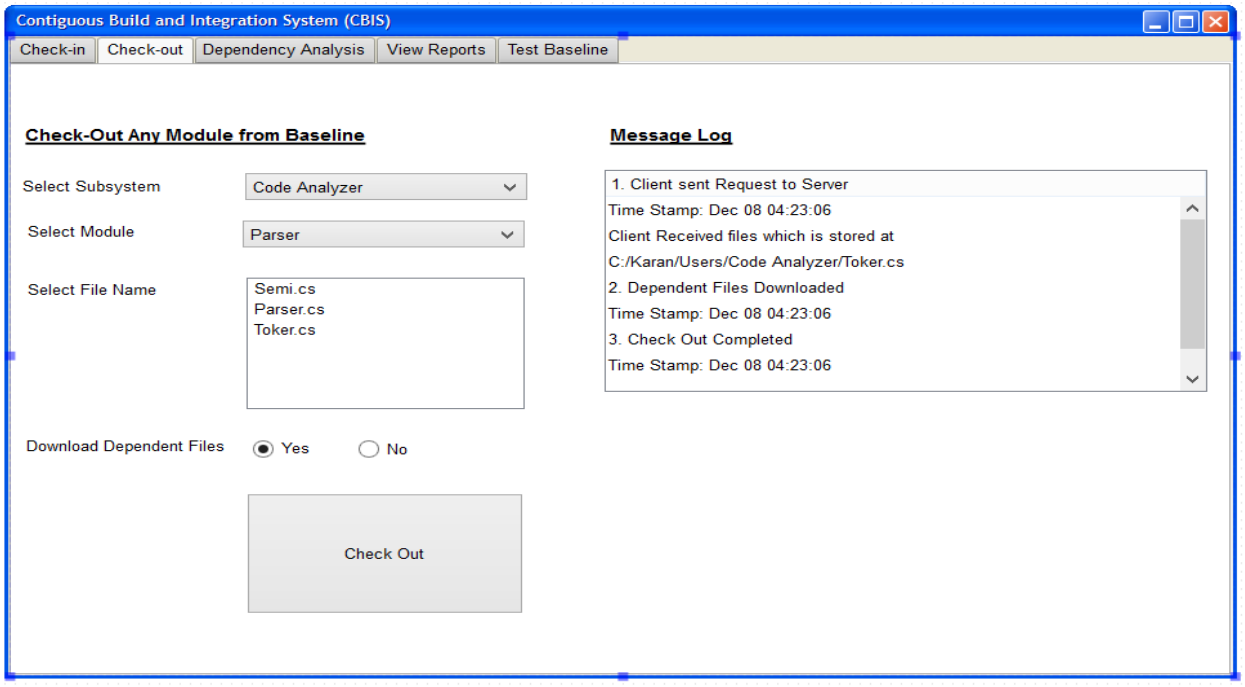
Figure 9 : Check-Out View of Client

Any user can check-out any module from the source code repository by selecting specific subsystem. After selecting subsystem, it will show Modules that resides within that subsystem. After selecting module, File Names will be displayed. We can choose which file/files we need to check-out.

User can download dependent files of selected files also by checking radio button yes. At the end by clicking Check-Out button client will send request to Server for check-outing the specified files.

5) Check-Out View of Client with Message Logs

After Check-out request is made by the Client, client waits till result receives. In order to do so, Message Log has been maintained to notify Client about which background activity has been completed.



Select Subsystem: Code Analyzer

Select Module: Parser

Select File Name: Semi.cs, Parser.cs, Toker.cs

Download Dependent Files: [✓ Yes] No

```
1. Client sent Request to Server
Time Stamp: Dec 08 04:23:06
Client Received files which is stored at
C:/Karan/Users/Code Analyzer/Toker.cs
2. Dependent Files Downloaded
Time Stamp: Dec 08 04:23:06
3. Check Out Completed
Time Stamp: Dec 08 04:23:06
```

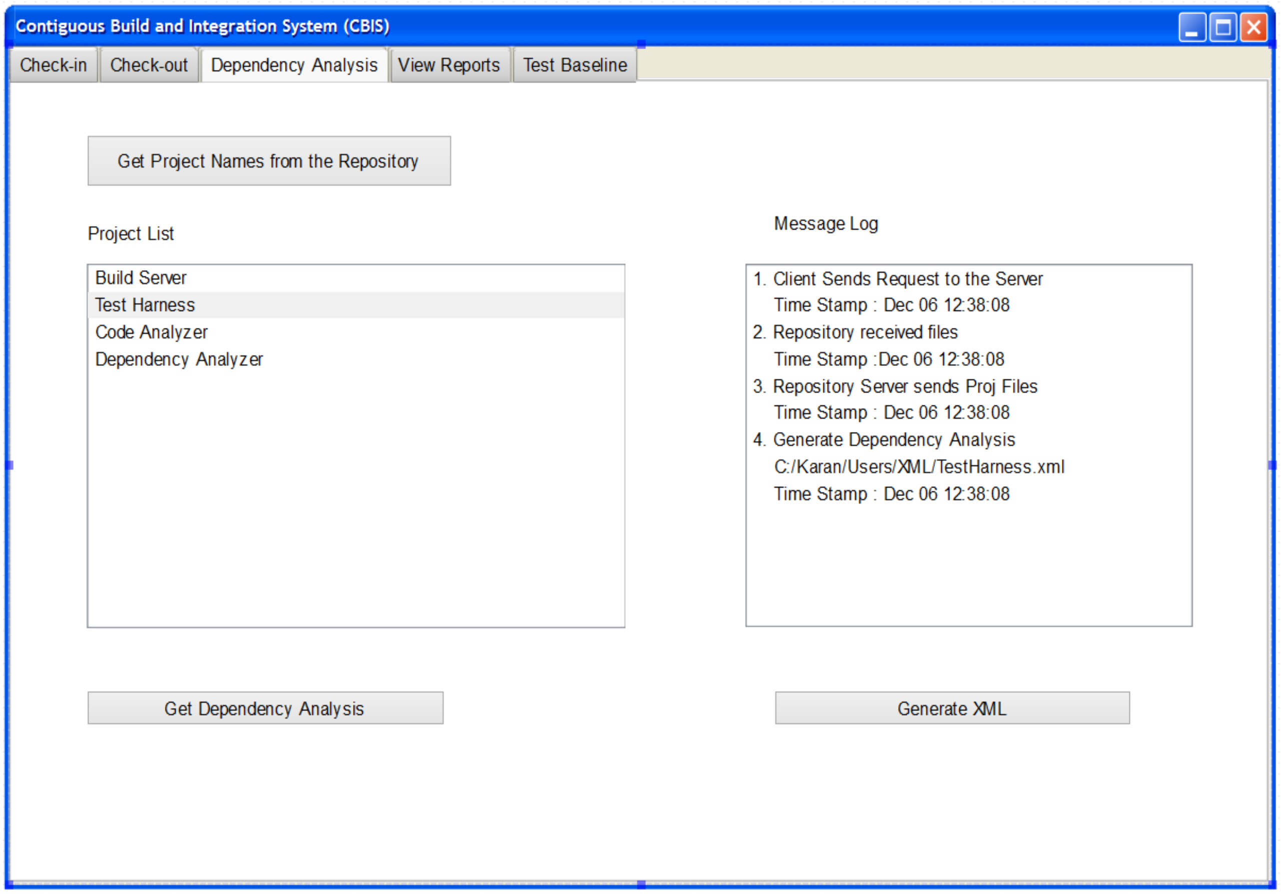
Figure 10 : Check-Out View of Client with Message Log

Time Stamps are used to let Client know about the timings of the activities completed at the server side. Thus, Message Log is very helpful for the client. The possible message log that may displayed during Client-Server Communications are:

1) Client Sends Request to Server.
2) Client Failed to make Request to Server.
3) Client Receive Requested Files.
4) Client Failed to Receive Requested Files.
5) Dependent Files Downloaded
6) Failed to download Dependent Files
7) Check-Out Completed.
8) Check-Out Failed.

6) Dependency Analysis View of Client with Message Log

This functionality offers to find dependency of the selected projects. This function is very useful for Software Architect and Tester. Software Architect must know the dependencies between the projects, then after only he can prepare document for enhancing the present system.



```
1. Client Sends Request to the Server
   Time Stamp : Dec 06 12:38:08
2. Repository received files
   Time Stamp :Dec 06 12:38:08
3. Repository Server sends Proj Files
   Time Stamp : Dec 06 12:38:08
4. Generate Dependency Analysis
   C:/Karan/Users/XML/TestHarness.xml
   Time Stamp : Dec 06 12:38:08
```

Figure 11 : Dependency Analysis View of Client with Message Log

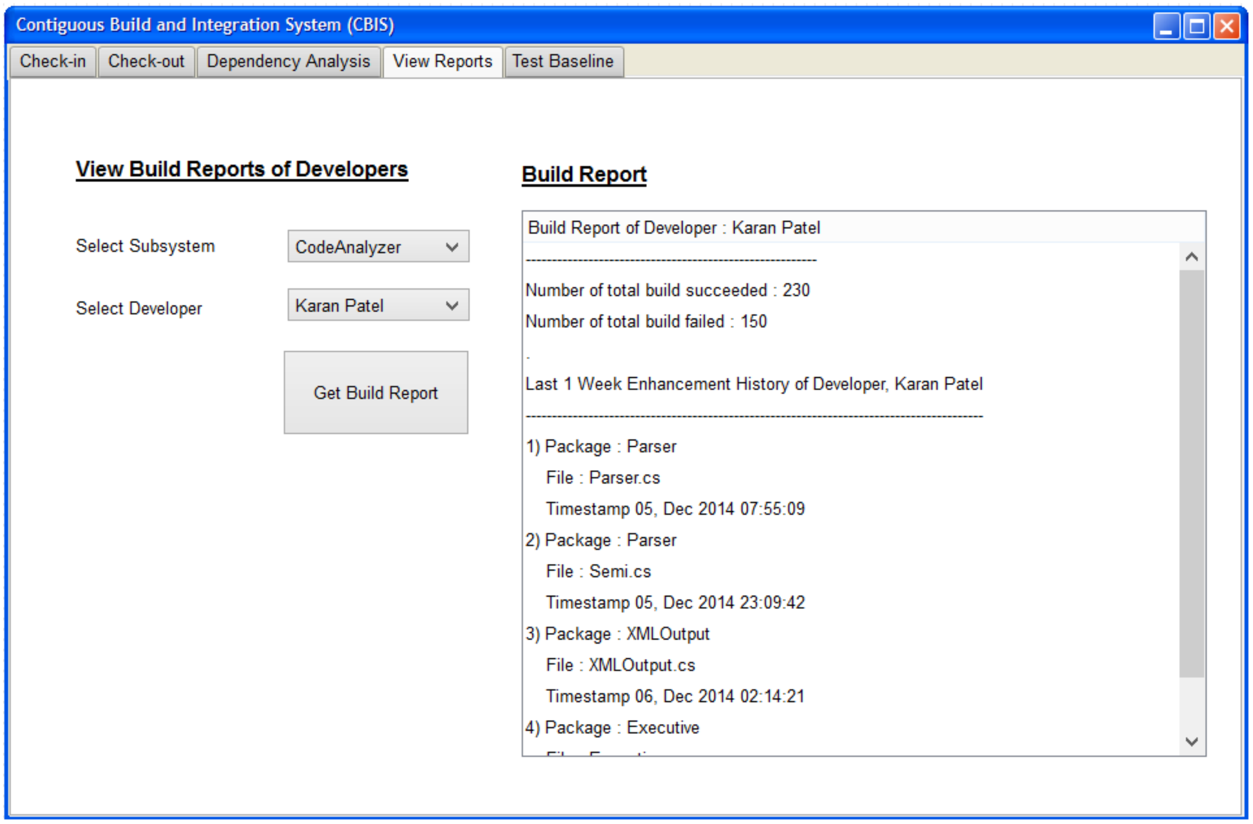
Source Code Repository works as storage of all the projects. By clicking the Get Project names from the Repository button, we get all Project names that resides at the baseline along with all the versions. Client can select any one or more Projects among the displayed list to get dependency analysis.

After making request from the Client, User can download the Dependency Analysis result by clicking Generate XML button. File location will displayed in the Message Log.

7) View Report View of Client

This is the view through which any user can see the Build Report of the Source Code Repository. This view is restricted to only some of authorized users. This view can be access by only following users.

- Project Manager
- Team Leader
- Software Developer



```
Build Report of Developer : Karan Patel
------------------------------------------------
Number of total build succeeded : 230
Number of total build failed : 150
-
Last 1 Week Enhancement History of Developer, Karan Patel
---------------------------------------------------------------------------
1) Package : Parser
   File : Parser.cs
   Timestamp 05, Dec 2014 07:55:09
2) Package : Parser
   File : Semi.cs
   Timestamp 05, Dec 2014 23:09:42
3) Package : XMLOutput
   File : XMLOutput.cs
   Timestamp 06, Dec 2014 02:14:21
4) Package : Executive
```

Figure 12 : View Report View of Client

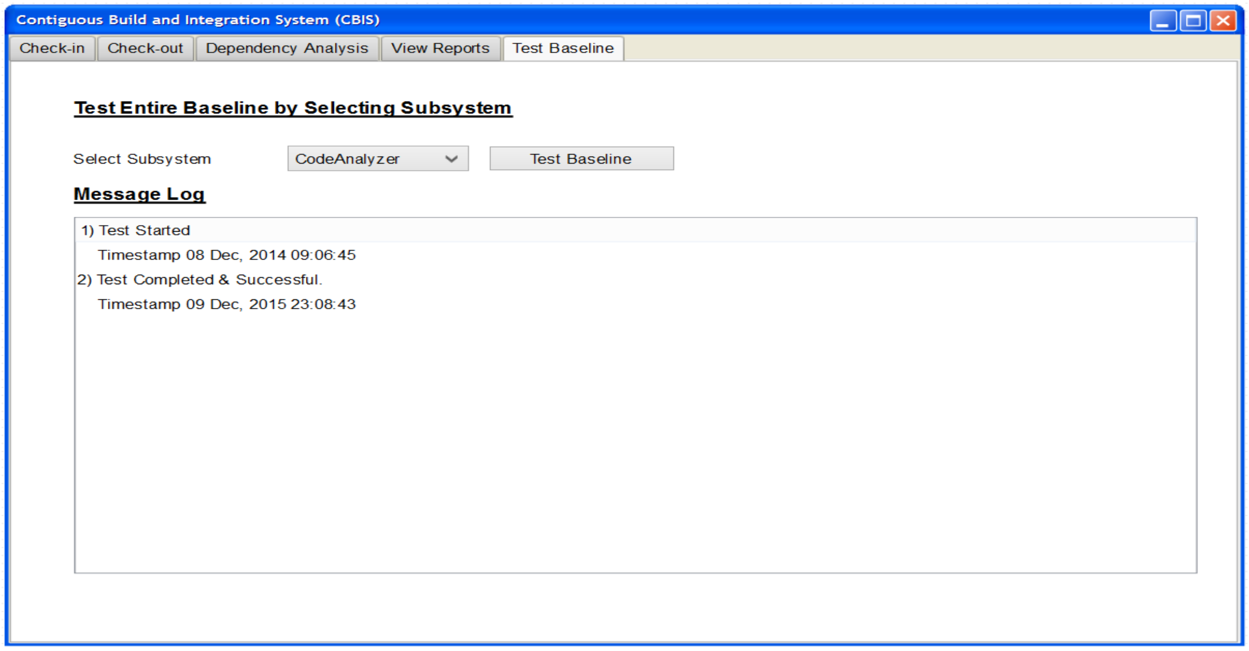
Team Leader/Project Manager periodically sees the performance of the each of the developer by accessing this functionality. Team Leader/Project Manager can select any subsystem and developer to evaluate his performance and to know any enhancement or modification done by him.

Build Report Includes the Total no. of Builds succeeded and Total no. of Builds failed of selected developer. In addition to this, Last 1 week enhancement history of that developer is also displayed. This build report is very useful to evaluate the performance of any developer.

8) Test Baseline View of Client

This view is accessible by Tester, Project Manager & Team Leader. Most of the time maintenance task has been carried out by testing entire subsystems that resides within the baseline. This view provides the option for testing the subsystem.

Message Log is maintained here to display the output of the testing. If test gets successful then it will display Test Completed & Successful else it will provide a detailed log of where test fails and which functionality is missing there.



```
1) Test Started
   Timestamp 08 Dec, 2014 09:06:45
2) Test Completed & Successful.
   Timestamp 09 Dec, 2015 23:08:43
```

Figure 13 : Test Baseline View of Client

4. REPOSITORY SERVER STRUCTURE

In order to understand the tasks and the activities carried out at the Repository Server, the Repository Server subsystem is divided into following module.

4.1. Repository Server Overview

The repository server is the heart of CBIS system where entire data of the project is located. It provides user with the facility to insert and extract the code from the baseline. Repository Server stores all developing baseline. Every module that tested correctly by the Test Harness Server will be added to the baseline. If module is not stored in the repository then it is not part of the baseline.

Checking-in and Versioning of the modules are the most important tasks of Repository Server which we will discuss later. It also maintains metadata for the files and modules in the baseline. It also carries out the dependency relationships for the incoming code.

Based on the request of the Build Server for the source code for the testing that is needed by the Test Harness Server, The Repository Server provides requested source code along with the mapping file of corresponding Test Cases to the Build Server.

The Repository Server is responsible for adding fetching and adding new module to the baseline. Versioning and Mapping are the two important policies that proposed here. If Client tries to add new module to the baseline, Repository Server maintain the versioning and meta data regarding all new check-in requests. Repository Server also provides all these information to the user while check-outing.

Main functions of Repository Server include:

- Stores modules as a collection of packages, generally it refers as baseline.
- Supports check-in, check-out, extraction, and versioning of all modules.
- It adds orphan packages to the baseline, which are not currently part of the module. It happens only after if test cases provided by the client testified.
- Allows extraction of modules for the Client and maintains a record of checked-out modules.
- Manages meta data of all modules that are checked-in and checked-out.
- Manages and stores the test suits source code which are needed by the Test Harness Sever.
- Finds dependent packages by executing queries about dependency analysis based on the check-in and check-out request.
- It provides result to the Client regarding new module has been added to the baseline or not.

- Return the Result Log to the Client via appropriate notifications.
- Identifies Clients who have access to check-in any modules into the baseline.
- Generates XML file that represent the dependent packages of the requested module.

4.2. Repository Server Context Diagram

The context diagram of the Repository Server is as follows:

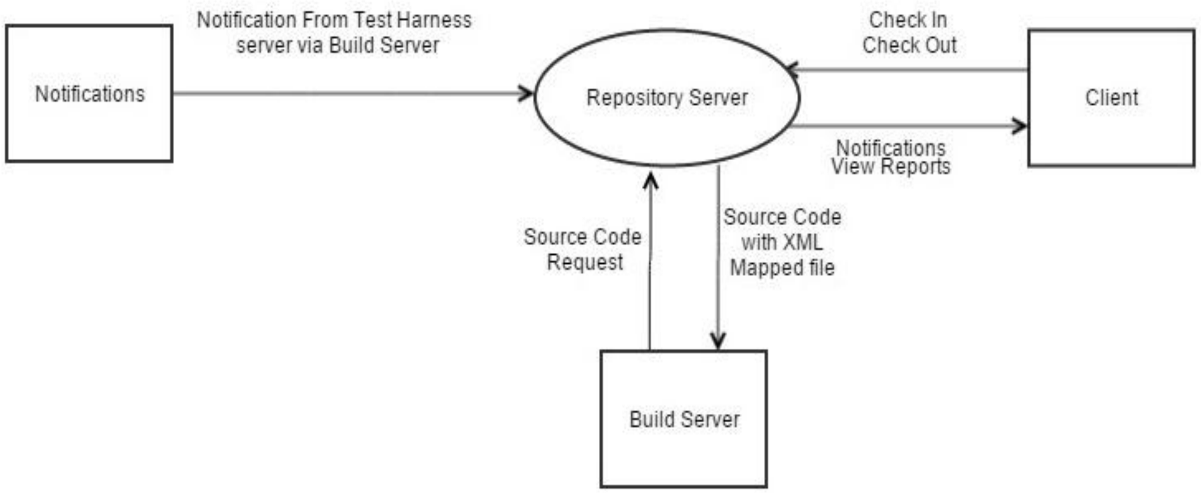


Figure 14 : Context Diagram of Repository Server

Context diagram represents the interaction of the Repository Server with the other subsystems. Build Server and Client are the two subsystem to which Repository Server interacts. Repository indirectly interacts with the Test Harness Server through Build Server.

Repository Server handles Check-in and Check-out request from the client. Based on the request it performs task. In case of module check-out request it authenticate the Client and give requested module along with the notification, versioning and meta data report. For check-in request it gets source code request from the Build Server simultaneously. It provides source code with XML mapping file to the Build Server. At the end of the testing at the Test Harness Server, Repository gets notification via Build Server regarding adding or dumping new check-in module to the Baseline.

4.3. Policies for Repository Server

1) Versioning

We have maintained our Repository Server as an immutable. Our fundamental idea is to test and integrate the module. So it may possible that there will be requirement of Versioning. This will help us in rolling back to the previous version in case the enhancements in the current version get failed. The process of versioning is as mentioned below, and sample of version tree is demonstrated in figure.

After receiving test passed notification from the Test Harness Server, Repository Server will have two options:

- The module which is to be added is having a copy of it inside the baseline. In this case the version number just increments but the previous copy also remains in the baseline.

- If the module name is not available in the XML file in the meta data which implies the package is checked in for the first time, then an entry is inserted in the XML file for the given file with a default version 1.0.

Notations that are used in figure to describe versioning:

X.X : represents stable version

X.X.X : represents beta version (any unstable version that is going for further testing soon)

•

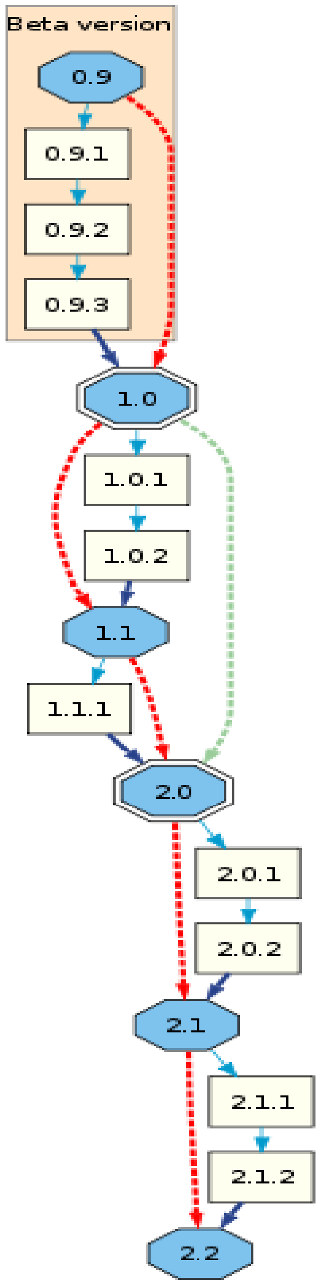


Figure 15 : Software Versioning

2) Meta Data Storage

Mata Data contains all the data regarding to the each subsystems and also maintain build maintenance history. These data is very useful for both analysis purpose and maintenance purpose. The Meta Data makes sure that every files even the duplicate ones (files with minor enhancements) can be distinguished from each other.

Meta Data stores the following information for every file:

- Owner of the file: use to follow authentic check-in
- Date, Time, Author and Version number: Use to identify each file by versioning and when it is created and by whom it is created.
- Number of times the build has succeeded or failed from any developer: Through which Team Manager/Team Leader get idea of the performance of the developer.

3) XML Mapping

XML Mapping is a very good policy to ensure that correct test case is correctly assign to particular file. Without XML mapping it's almost impossible to identify which test cases are needed by which files. This way Test Harness Server will know about the files and corresponding tests.

XML Mapping is done in two ways:

1) At the time of the check-in request, Client has to provide the module along with the test suits. All of these modules and test suits are then passed to the Repository Server. There is a requirement of mapping the particular file with the appropriate test case. So that mapping is done by the XML Mapper package by creating XML mapper file to ensure that the correct test case will be assign to the source code file.

2) When the Build Server requests for Source code along with the dependent code, Repository Server will map these code with correct test cases and will generate XML mapped file. These source codes, test suit and XML Mapped file will be needed by the Test Harness Server for testing purpose.

4.4. Repository Server Package Diagram

Following Package Diagram shows the Modular Structure of the Repository Server Subsystem. It suggests the internal flow of the Repository Server.

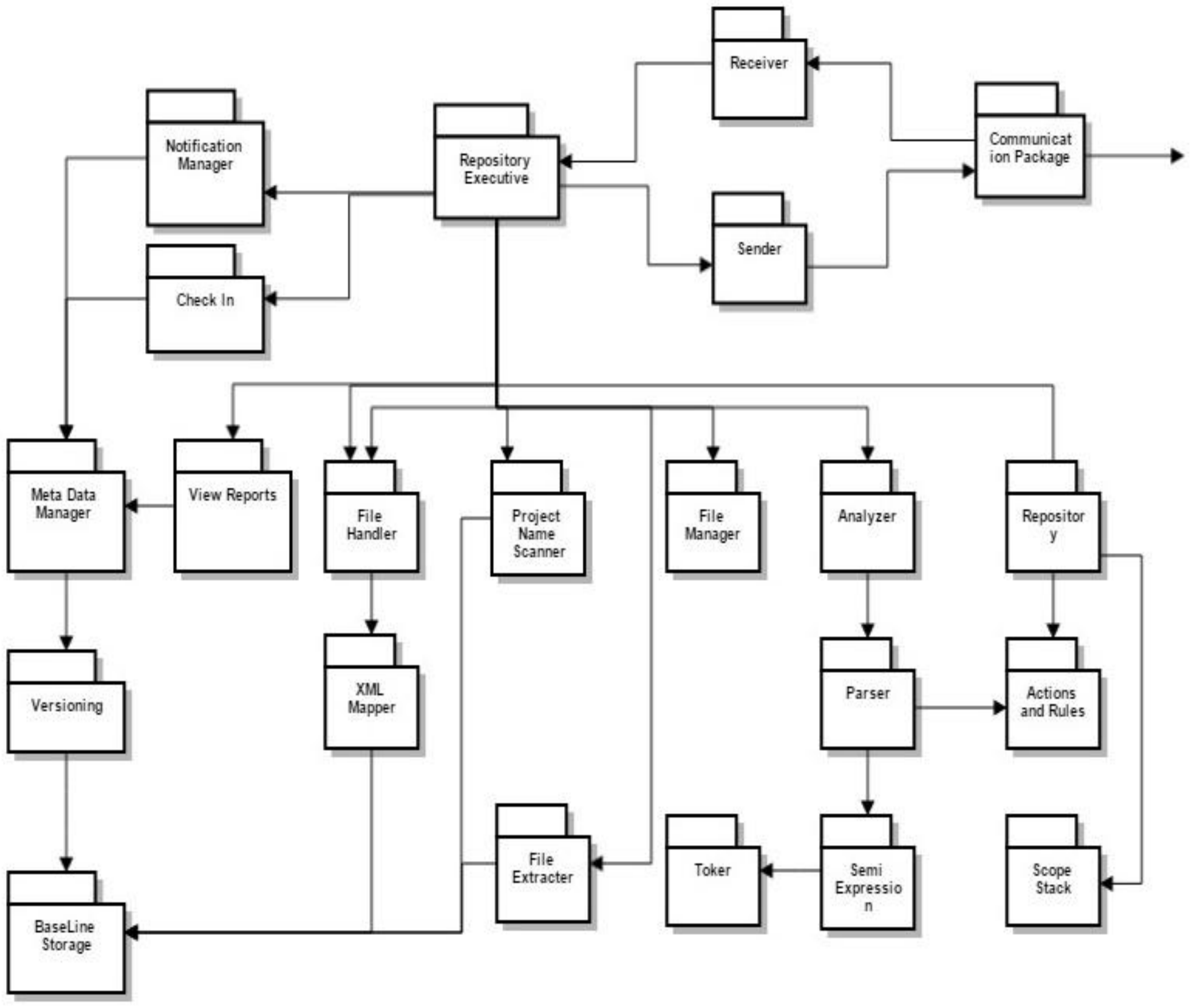


Figure 16 : Package Diagram of Repository Server Subsystem

Repository Server Executive Package:

Server Executive Package controls the entire flow of Repository Server. It accepts input from the Communication Package which is send by the Client and responsible for calling other packages at appropriate time. All the packages in the application are accessed via the Repository Server Executive Package. It is responsible for the working of the entire system. The important functionalities are discussed below:

It will serve Check-in request of Client by passing Client's request from Communication Package to the Check-In package. Before passing it to the Check-In package, it will use services of Project Name Scanner package to get list of projects and versions available at the baseline.

It will serve Check-out request of Client by passing Client's request from Communication Package to the File Extractor package which returns the Source Code of the projects required by the Client.

When the Client requests to do dependency analysis, the message is passed to the file manager which extracts the files and then those are passed to the Analyzer Package for further analysis.

It also serves the request from the Build Server. When any request comes from Build Server, it goes through this package and fetches the data from the File Handler Package.

Project Name Scanner Package

This package collecting all the names of the projects available. Whenever the server starts, it will check each project. After generating list of available projects it will transfer it to the XML encoder to convert into the XML file and stores it with the help of XML File Manager package. So now if clients requests again, we can send this XML file instead of checking all available project again.

Check In Package

Check-In package is responsible for serving the check in request of the Client. It receives Client's request from Executive package. It passes this request to the Meta Data Manager package which includes the task of checking the ownership, versioning. It ownership found then it passes this request to Analyzer package through Executive package for the task of Dependency Analysis.

Meta Data Manager Package

Upon receiving the request from the Check-In package, Meta Date Manager package is responsible for storing all the Meta data of the baseline files. It passes user's Check-in request to the Versioning package for proper versioning. It holds all the revision, versioning and modification history of every subsystem.

Versioning Package

Versioning Package receives the request from the Meta Data Manager to serve proper versioning for the each module that are going to add into the baseline. This package is responsible for maintaining the versioning of the baseline files.

Baseline Package

The baseline package is the very important package of our system. Source Code Repository is stored here in this package. It consists of immutable source code which is compiled and tested whenever there's any check-in request. The source code resides over Baseline is stable and never crashed. It maintains versioning of each of the module with the help from Versioning Package. The data which is already present in the base line cannot be changed it can just be versioned. It is called by many related Repository Server packages to retrieve the Source Code needed for the check-out or testing purpose for the Test Harness Server. This package works as a large Source Code Repository.

File Manager

File Manger package is responsible for extracting the files from the directory based on client’s request. These extracted Retrieved files are then transferred to Analyzer package for Dependency Analysis. The File Manager also receives request when entire baseline is to be analyze.

Analyzer Package

Analyzer package is very useful package for the formation of type table. Analyzer gets files from File Manager package. It uses the functionalities of Parser & Semi packages to create the type table.

Parser Package

Upon receiving the file stream from File Manager package, Executive passes it to the Parser package. Parser package has some specified rules for detection of grammatical syntax, generally known as containers of actions. For Example, non-keyword name comes before pair of brackets before opening curly bracket suggests the Function Name. Parser package uses the services of Semi & Toker to find out Function Name, Function Size & Complexity.

Semi & Toker Package

Semi package has Semi class that generates token-sequence based on specified rules (semiExpressions). semiExpression is a sequence of tokens that end in "{" or"}" or";" Toker package have a Toker class that reads tokens from the file stream. Toker provides, via the class CToker, the facilities to tokenize ASCII text files. That is, it composes the file's stream of characters into words and punctuation symbols. Toker will decomposed the file stream by removing comments and spaces, and then it passes to the Semi. After receiving decomposed stream from Toker package, Semi will return this meaningful decomposed stream to Parser.

Action and Rules, Repository Package

Action And Rules package is useful for calculating & manipulating the stream it receives from the Parser package. It mainly composed of rules and specified actions for specific rules. It uses the Repository Package as the intermediate container to store the result after performing required operations. For Example, as soon as Action and Rules Package gets the function name, function size, complexity & relationships, Repository is the Package where all these information will be stored for later uses such as querying the baseline as Build Server requests data.

Scope Stack Package

Scope Stack package provides, the facilities to track position in code scope by pushing and popping type, name, and place. The type might be a namespace, class, struct, function, or control. The name is the namespace of the instance of that type, and place is the line number in a package being analyzed.

File Extractor Package

File Extractor package takes check-out request from the Repository Executive package and pass it to the Baseline package in order to extract the entire source code file from the Baseline package. Baseline package will provide copy of the files needed by the Client and this package is passing it to the Client via Server's Executive package.

File Handler Package

The File Handler package is invoked by the Executive package upon receiving the request from the Build Server to provide source code along with the test suit. File Handler package then pass this Build Server's request to XML Mapping package to generate mapped XML file.

XML Mapping Package

XML Mapping package is invoked by the Build Server's request . It receives source code and test suits from the Repository through File Handler Package. These files and test suits are mapped here and it generates mapped XML file so that it can be given to the Build Server for compiling purpose which is useful for the Test Harness Server to carry out tests file by file. It receives request from the File Handler package to generate mapped file.

Notification Manager Package

Notification Manger is used to give a notification to the user about completion of the check-in or check-out activities. When all the test suits successfully tested, this package receives notification regarding to add module to the baseline by proper versioning. test Harness Server sends Notification to the Repository Executive for adding module to the Baseline or not. After adding it to the Baseline, it sends notification to the user regarding check-in completion. Build server is also sends notification to the Repository Executive package regarding requesting of source code along with XML mapped file.

View Report Package

Whenever Client makes request of getting Build Report of any subsystem, Repository Server passes this request to the View Report Package. It simply gets all of the information from the Meta Data Manager package and sends it back to the Client via Executive Package.

4.5. Repository Server's Activity Diagram

The Activity diagram represents the graphical representation of the entire system. It represents the major activities that should happen during the execution of entire system.

1) Check-in Activity

The flow of the check-in activity is discussed below.

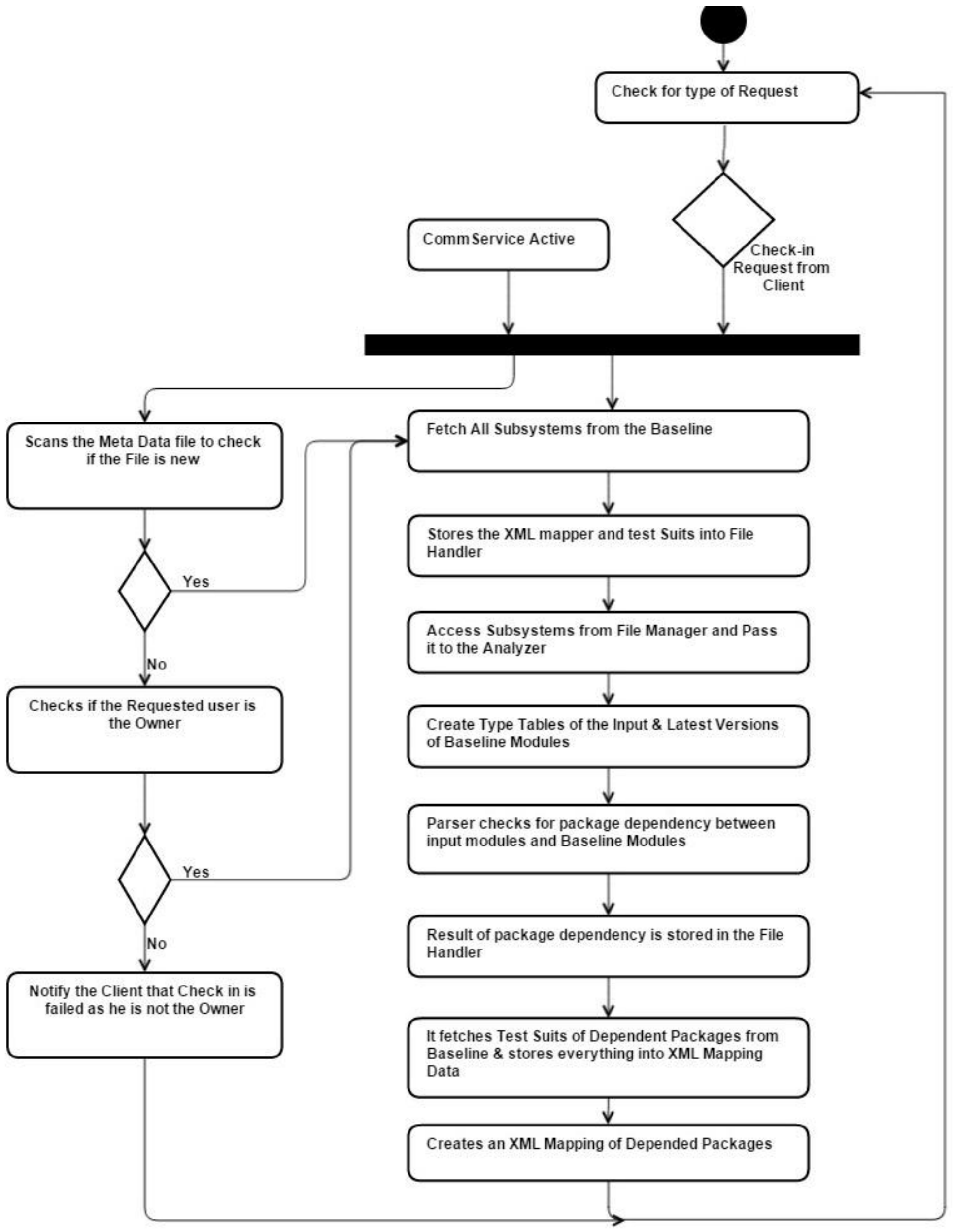

Check for type of Request
Comm Service Active
Check-in Request from Client
Scans the Meta Data file to check if the File is new
Yes
No
Checks if the Requested user is the Owner
Yes
No
Notify the Client that Check in is failed as he is not the Owner
Fetch All Subsystems from the Baseline
Stores the XML mapper and test Suits into File Handler
Access Subsystems from File Manager and Pass it to the Analyzer
Create Type Tables of the Input & Latest Versions of Baseline Modules
Parser checks for package dependency between input modules and Baseline Modules
Result of package dependency is stored in the File Handler
It fetches Test Suits of Dependent Packages from Baseline & stores everything into XML Mapping Data
Creates an XML Mapping of Depended Packages

Figure 17 : Flow of Check-in Activity

Check-in Activity Description

Once Client requests the check-in request from the GUI, the following events take place at Repository Server.

- Receives the Client's check-in request by accepting the modules and test suit provided with it.
- Meta Data Manger will check if check-in file is new or not.
- If it is new then it will start fetching all subsystems from the Baseline.
- But if it is a modified file then it will check the owner of that file and only allows file to be check-in if requested user is its owner.
- Other than above mentioned two activities, for all possibilities, Repository Server will notify Client regarding failed check-in.
- Repository Server then finds the dependent modules that are dependent on the provided module by using dependency analyzer.
- Analyzer will create the Type table of the input and latest versions of the baseline modules.
- Result of dependency analyzer is stored into the File Handler.
- Repository Server will create mapped XML file that maps Source code of file to the test cases provided by the users.
- Repository Server will hold this mapped XML file along with the source code, so that when Build Server requests for it, Repository Server will give it to Build Server, which will compile that source code and send it to the Test Harness Sever for testing procedure.
- The failed/passed testing result from the Test Harness Server will decide whether to check-in the package to the baseline or not.

2) Check-In Notification Activity

The flow of the notification activity is discussed below.

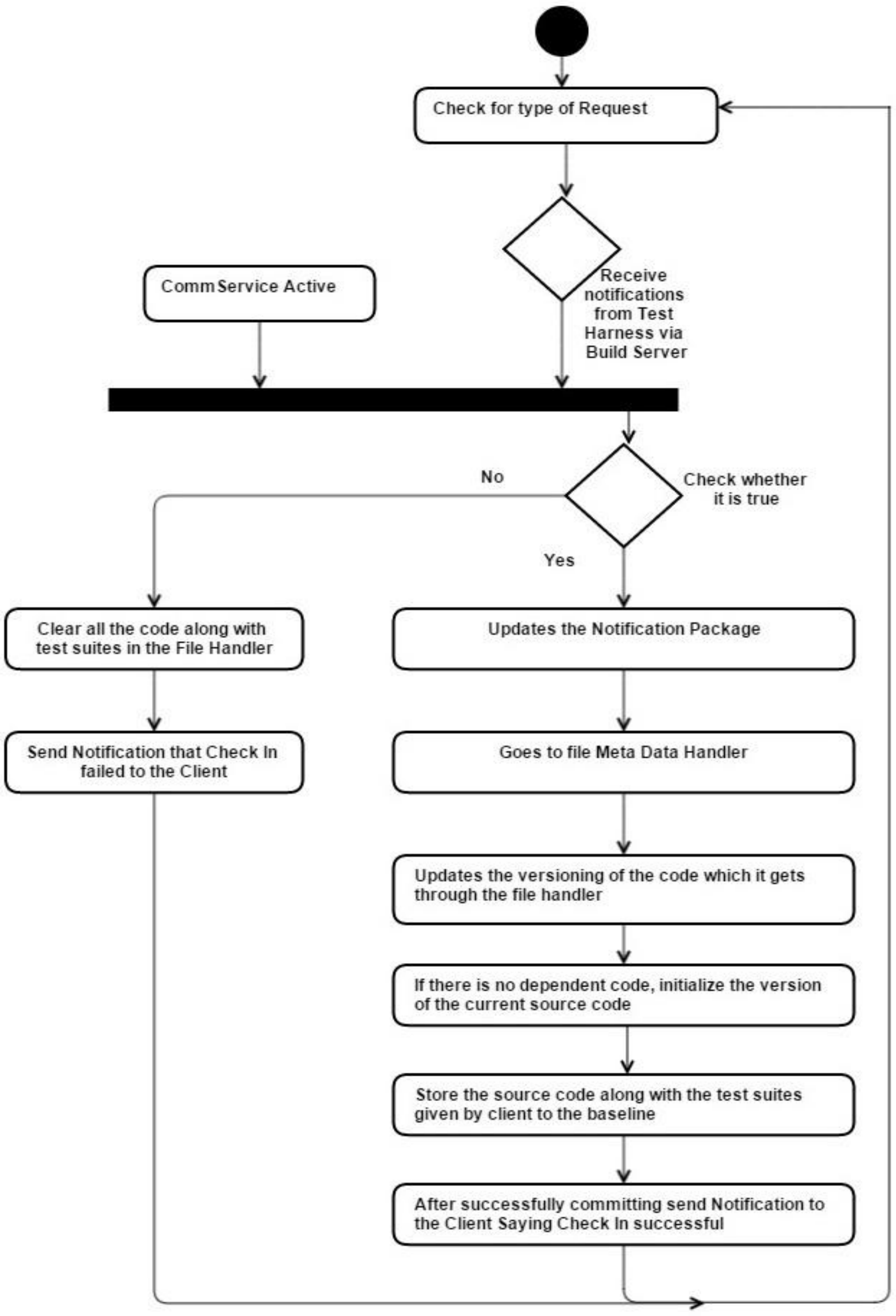

Check for type of Request
Comm Service Active
Receive notifications from Test Harness via Build Server
No
Check whether it is true
Yes
Clear all the code along with test suites in the File Handler
Send Notification that Check In failed to the Client
Updates the Notification Package
Goes to file Meta Data Handler
Updates the versioning of the code which it gets through the file handler
If there is no dependent code, initialize the version of the current source code
Store the source code along with the test suites given by client to the baseline
After successfully committing send Notification to the Client Saying Check In successful

Figure 18 : Flow of Check-In Notification Activity

Check-In Notification Activity Description

Once Reposritry Server sends required source code along with the test suit XML Mapped file. It waits till the testing at the Test Harness Server completed. Once testing is done, test Harness Server sends notification to the Repository Server via Build Server.

Events take place at Repository Server side are:

- It will interpret the incoming notification message.
- If testing is failed then Repository Server clears all the code along with the test suits provided by the Client from the File Handler.
- And Send failed check-in notification to the Client.
- If testing succeed, then Repository updates its own Notification Package.
- Next, Meta Data Handler updates the versioning of the code of the current source code get through File Handler.
- Repository Server then stores the source code along with the test suits to the Baseline. And after successful commiting, it will notify user regarding succeesful check-in.

3) Check-out Activity Description

The flow of the check-out activity is discussed below.

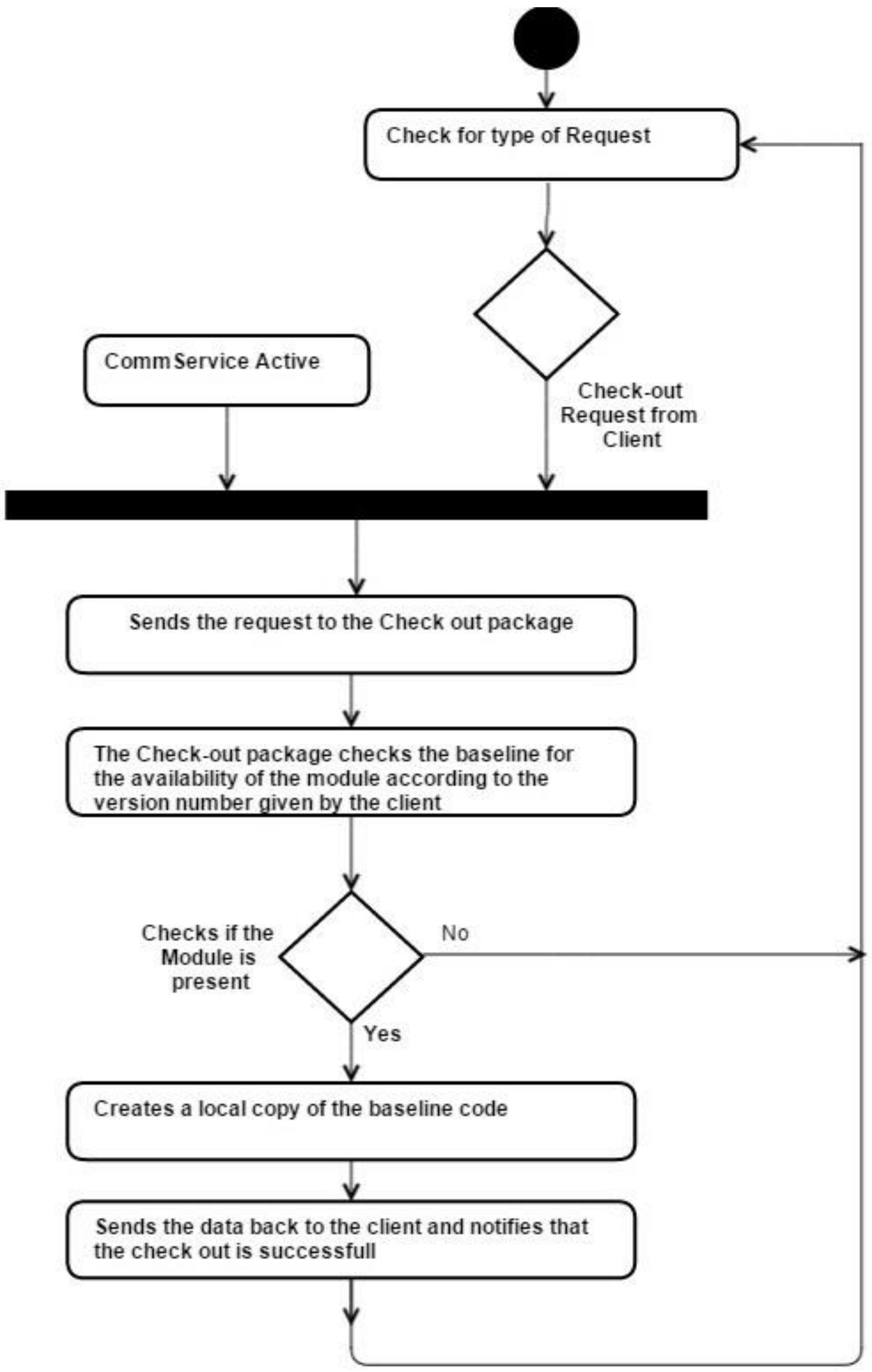

Check for type of Request
Comm Service Active
Check-out Request from Client
Sends the request to the Check out package
The Check-out package checks the baseline for the availability of the module according to the version number given by the client
Checks if the Module is present
No
Yes
Creates a local copy of the baseline code
Sends the data back to the client and notifies that the check out is successfull

Figure 19 : Flow of Check-out Activity

Check-out Activity Description

For the module Check-out functionality, any user can download it by providing authentic credentials. User has to provide which module he wants to check-out along with the version number.

If user selects Check-out activity from the GUI, the following events take place at Repository Server.

- To handle check-out request, Repository Server will send this request to the File Extractor/Check-Out package.
- The File Extractor package first checks the availability of the requested module by checking its name and version into the baseline.
- If module is not present then, it sends notification message to the Client regarding failed check-out.
- If requested module is present at baseline, then File Extractor package generates the local copy of it.
- Sends this local copy to the Client and notifies client regarding Check-out successful.
- Sometimes Client requires dependent modules also be check-out along with the requested module. In this case, all of these modules are being fetched from the baseline.

4) Dependency Analysis Activity

The flow of the dependency analysis activity is discussed below.

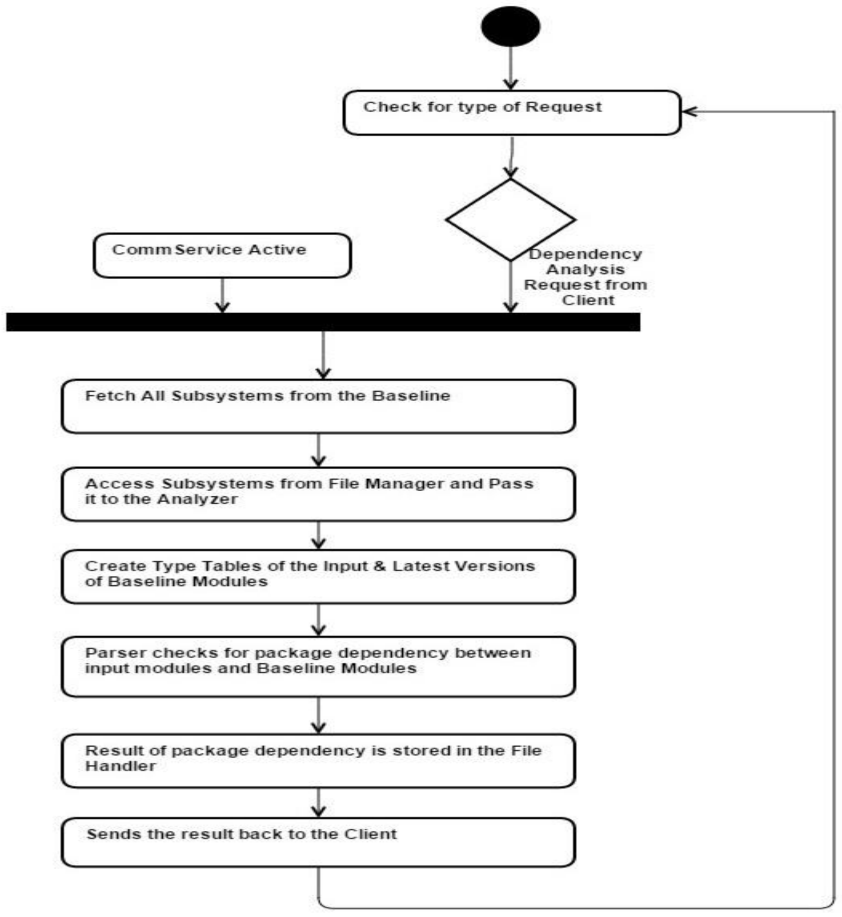

Check for type of Request
Comm Service Active
Dependency Analysis Request from Client
Fetch All Subsystems from the Baseline
Access Subsystems from File Manager and Pass it to the Analyzer
Create Type Tables of the Input & Latest Versions of Baseline Modules
Parser checks for package dependency between input modules and Baseline Modules
Result of package dependency is stored in the File Handler
Sends the result back to the Client

Figure 20 : Flow of Dependency Analysis Activity

Dependency Analysis Activity Description

Software Architect may require to find only dependency of the selected subsystems. Events take place at Repository Server are as follows:

- Repository Server notifies File Manager to fetch all the sysbsystems and pass it to the Analyzer. Analyzer will create Type Table using the functionalities of Parser & Semi and checks for dependency of that module.

- Result of the package dependency is stored into the File Handler. And It sends back to the client as an XML file

5) View Reports Activity

The flow of the view report activity is discussed below.

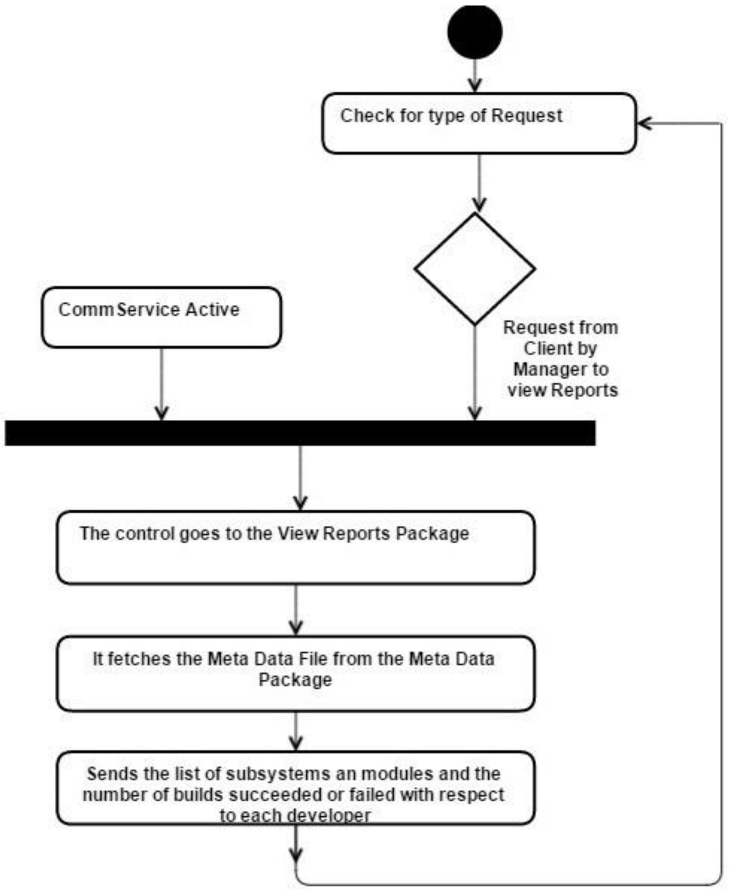

Check for type of Request
CommService Active
Request from Client by Manager to view Reports
The control goes to the View Reports Package
It fetches the Meta Data File from the Meta Data Package
Sends the list of subsystems an modules and the number of builds succeeded or failed with respect to each developer

Figure 21 : Flow of View Report Activity

View Report Activity Description

Events take place at Repository Server are as follows:

- As soon as Project Manager/Team Leader request for View Report of the development activities, the control goes to the View Report package and fetches the meta data file from the Meta Data Package.

- By analyzing meta data file, it fetches useful statistics about the developer, regarding number of builds failed or succeeded.

- It also fetched 1 week enhancement history of that developer.

- All of these information has been send to the Client in the form of the Build Report.

5. Test Harness Server

Testing and integrating our code into the Repository Server is core task of CBIS. Test Harness Server provides simultaneous testing facility to the CBIS.

5.1. Test Harness Server Introduction

Test Harness Server provides tool that supports frequent testing of the files with the test suit. It is an automated tool saves a lot of time of any tester if he's working on a large project. Test Harness Server requires XML mapped file of Source Code with the test cases that is provided by the Repository Server through Build Server. This XML mapping files tells Test Harness Server to execute specific tests for specific files. The Test Harness provides a test-vector generator for programmed test inputs like messages, lines of text, files, etc and a logger for writing test reports. Test Harness Server activities are based on test configurations, each configuration runs its own thread. The Test Harness Server can also be used for unit testing, assembly testing, integration testing and regression testing.

Test Harness helps the Clients like software developers, test teams and managers to check the quality of the software and provides help to integrate the modules into the existing system by continuous testing of new modules that are going to be added. Below are the key uses of Test Harness Server

- It tests the new source code or modified version of existing source code file checked-in the Repository by running the appropriate test cases.
- Saves the test results, summary and logs at the Repository Server.
- Generates notification for the Repository Server regarding advance the check-in procedure to add module to the baseline.

5.2. Test Harness Server Context Diagram

The context diagram of Test Harness Server is explained below.

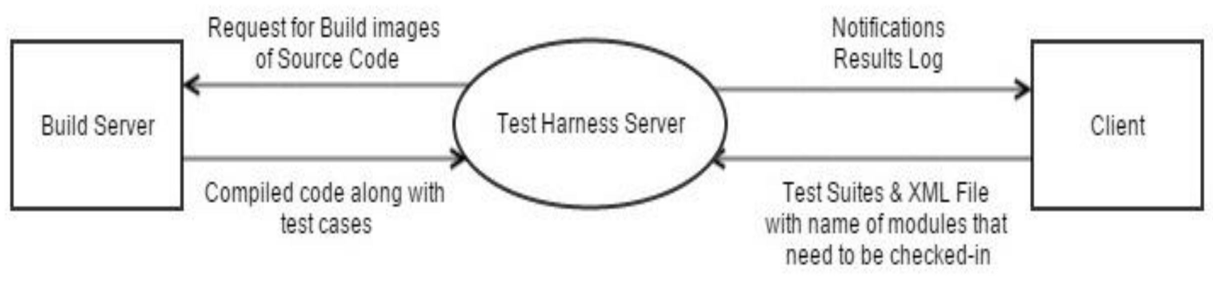

Request for Build images of Source Code
Notifications Results Log
Build Server
Test Harness Server
Client
Compiled code along with test cases
Test Suites & XML File with name of modules that need to be checked-in

Figure 22 : Flow of View Report Activity

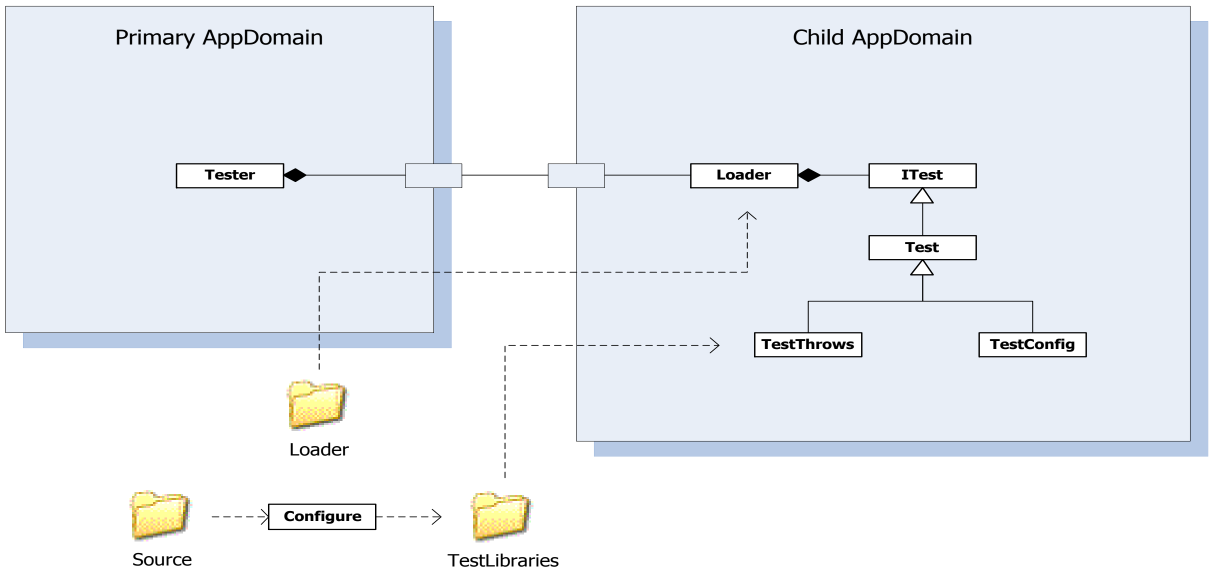
The main components of the Test Harness Server are the Build Server, and the Client. Test Harness Server indirectly communicating with the Repository Server through Build Server. When Client made request for Check-in, its request passes to both the Repository and test Harness Server.

When Test Harness Server receives the request from Client, it requests the Build Server to provide the build images of the source code along with the test cases. Build Server compiled the source code and test cases provided by the Repository Server and sent it back to the Test Harness Server for testing of new module.

The Test Harness Server sends the results back to the client in form of notification indicating if the tests were passed or not. It also sends notification signals to the Repository Server through Build Server for indicating the start of the Check-in procedure at baseline.

5.3. Test Harness Server Architecture

The below architecture diagram shows the architecture of the test harness server. Test server creates a new AppDomain to run tests. This is to make sure that any exception or crash in test code does not disrupt the test harness server.



SOURCE : http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/code/TestHarnessPrototype/TestHarnessPrototype.pdf

Figure 23 : Architecture Diagram for Test Harness

Test Driver creates a new App Domain to run tests. The advantage of creating a new App Domain is that even if any exception happens or it crashes, it does not affect the working of the entire test harness. As test harness does not have any human intervention, it is necessary to keep the server always up and running.

In the Child AppDomain server loads the build images of the source code,XML mapping files and test suites to runs the test. Test runner is aided by test vector generator and logger to log the test results. Tests can be achieved with test harness when child domain can run on threads. The threads keep running unless all the test data runs out. Number of parallel tests can be simulated by configuring as many threads as required.

5.4. Test Harness Server Policy

Cache

In order to maintain the performance of the CBIS, we proposed a policy of Cache. Test Harness Server needs build images in order to run the sequence of tests. In standard way whenever client request for check-in, test Harness Server requests the build server to make an build image. Build Server requires source code and the test suit in order to build build images of that code. So it request Repository Server to provide source code along with the test cases. For each check-in request Test Harness Server has to wait for this server-server communication, which takes longer time to execute tests.

We proposed Cache at the Test Harness side, which stores build images of recently obtained from the Build Server. So Test Harness Server only requests for the build images to the Build Server, which are not present into the Cache. The cache improves the performance of the CBIS system. Also the test harness does not have to wait for the files to arrive. It can start performing its tasks as soon as it finds the required data in its own cache.

5.5. Test Harness Server Package Diagram

Test Harness Server Package Diagram shows the Modular Structure of Client Subsystem.

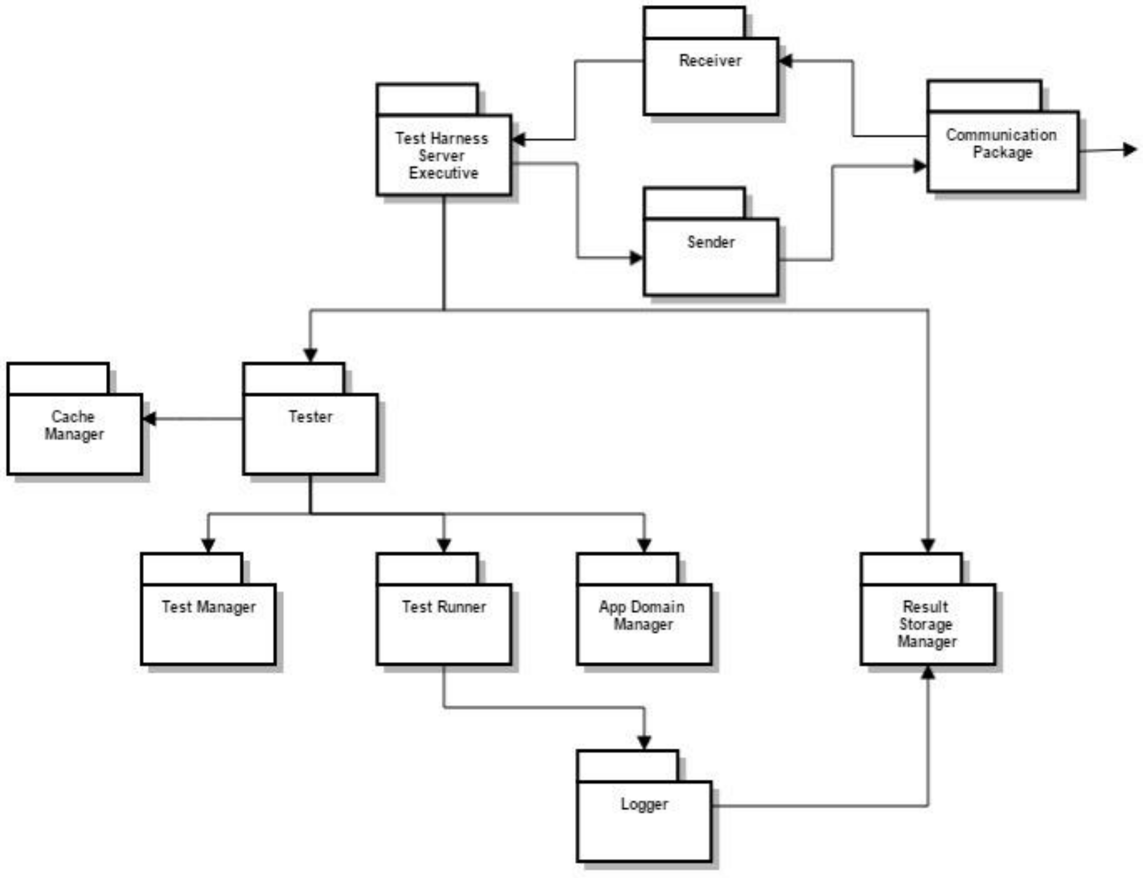


Figure 24 : Package Diagram of Test Harness Server

The Test Harness Server is connected with the Build Server and the Client via communication channel to exchange the request-response between these subsystems. All of the notifications of the result are going through this channel only. The functionalities of all packages of the Test Harness Server is explained below.

Test Harness Server Executive Package

It controls entire flow of this subsystem. This package is used by the subsystem to manage all the activities of the system such as loading all the test cases and executing them, notifying server, notifying servers etc. It takes care of any requests coming from the client or other servers. When a request arrives from the client for the testing of the code, this package passes it to the tester package.

Tester Package

Upon receiving the request from the Executive package, it checks if the required Baseline files are available in the cache. The tester package is responsible for checking if the required base line files are present in the cache or not by passing the module names into the Cache module. The main task of this package is running the tests and recording the test results. It utilizes the Thread Manager to run each test on an individual thread and uses test runner to run threads.

Cache Manager Package

This package is called every time by the Tester package to check whether Cache has the baseline files or not. If it's there then it returns back to the tester Package.

App Domain Manager Package

This package is responsible for creating and destroying child app domains for test to run. This enables test codes to be executed in an isolated memory space different from the main app domain.

Test Manager

After receiving the Build Image from Build server, Tester package passes all of these source code files, test suits along with XML mapper file to the test Manager Package. Test Manager works as a repository of all these intermediate data. After receiving all of these data it sends it to the Test Runner package to start tests on the module.

Test Runner

This package loads the test driver DLLs(test suits) and the source code executables from the Test Manager package. It starts running test upon loading of all files needed for tests and maintains logs of each test in the Logger Package.

Logger Package

This module is responsible for collecting all the test results. The test results can have errors, unexpected bugs, failures during the execution of tests. This module is responsible for representing errors and failures in different ways. This result has to be stored in the Result Storage Manager.

Result Storage Manager

All the test related results are stored here through Logger package. These test results will be useful to send notification to the Client and Repository server for the further processing.

5.6. Test Harness Server's Activity Diagram

The Activity diagram represents the graphical representation of the entire system. It represents the major activities that should happen during the execution of entire subsystem.

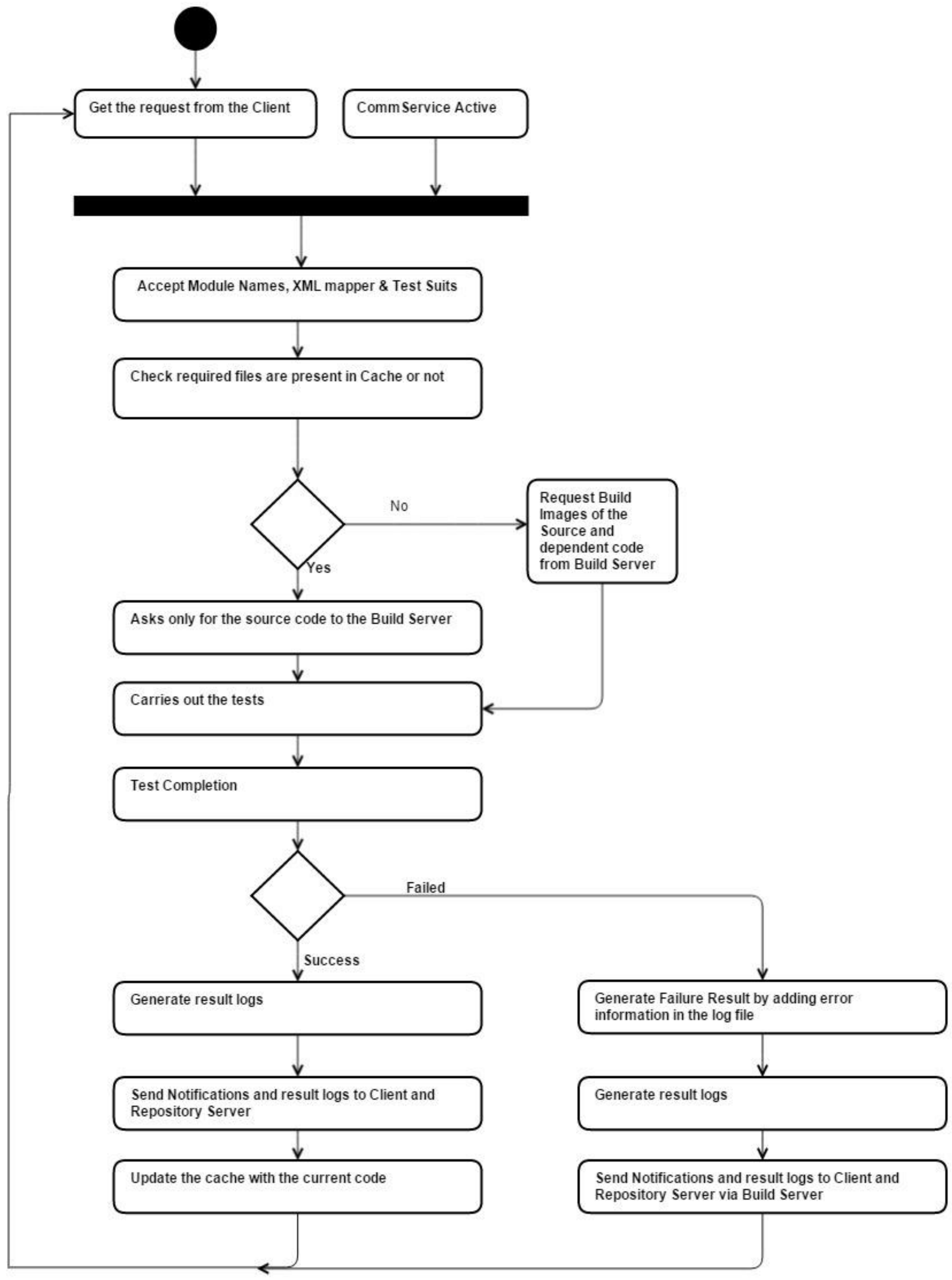

Get the request from the Client
CommService Active
Accept Module Names, XML mapper & Test Suits
Check required files are present in Cache or not
No
Yes
Request Build Images of the Source and dependent code from Build Server
Asks only for the source code to the Build Server
Carries out the tests
Test Completion
Failed
Success
Generate result logs
Generate Failure Result by adding error information in the log file
Send Notifications and result logs to Client and Repository Server
Generate result logs
Update the cache with the current code
Send Notifications and result logs to Client and Repository Server via Build Server

Figure 25 : Activity Diagram of Test Harness Server

Once Client make Check-in request the events that are going to take place at Test Harness Server side are explained below.

- Client made check-in request by sending Modules and test cases to the Test Harness Server. Test Harness Server accepts all these information.

- After receiving these modules along with the test cases, it checks with the Cache to make sure that if there's any Build Image for the same or not.

- If it is not present then Test Harness Server will request the Build Server to build images of source code and its dependent code. Else it will only request to make build image of source code.

- As soon as build images and XML mapper files are received by the Test Harness Server, it will start testing of the given module.

- If the tests during the testing fail it logs an error report into the logger. And then it notifies to the Client and also the repository regarding the failure via notification.

- Test Harness Server will not update the cache in case of the failure.

- If it is a success then the results of success are logged in the logger. The notification will be sent to the Repository server and Client regarding test has been executed.

- After the tests have been successful the test harness updates the cache with the latest code. This can be further used for the next request if it uses the same code which is stored.

6. Build Server

6.1. Introduction to Build Server

Build Server is the intermediate server between the Test Harness Server and Repository Server. Build server provides service to compile and build executables for. Build server uses a build script to build a library or execution image for a specified set of packages. Every source code checked-in to the Repository requires a build and if the build fails the check-in fails. Every test configuration also requires a build before executing tests in the configuration.

6.2. Build Server's Context Diagram

The context diagram of Build Server is as follows:

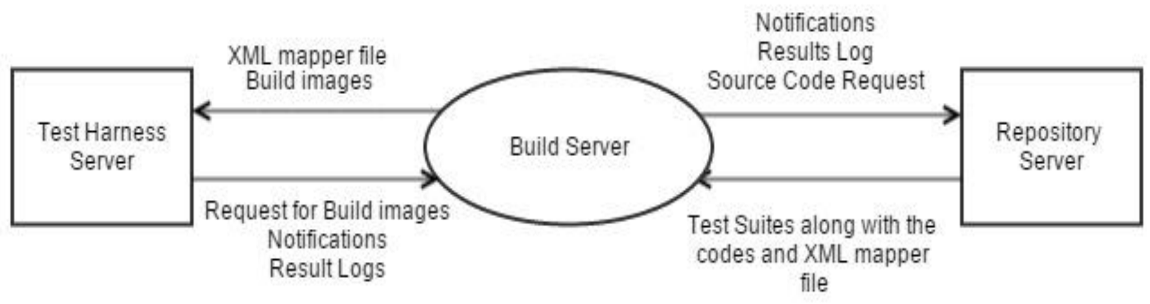


Figure 26 : Context Diagram of Build Server

The Repository Server and the Test Harness Server are the main components of the Build Server. The Build Server gets build image requests from the Test Harness Server. Upon receiving the request, Build Server first checks build image that might resides in the Cache memory. If it's not there then Build Server requests the Repository Server for providing the Test Suits along with the Source code. Repository Server will serve the source code, test suit and XML mapped file to the Build Server. Then Build Server compiles this source code to generate build image and forwards it to the Test Harness Server.

The Build Server also works as an intermediate between Test Harness Server and the Repository Server for forwarding the notification message between them. Notification message can be used to notify Repository Server to follow up Check-in procedure after testing completed.

6.3. Build Server Package Diagram

The package diagram of the build server is as follows:

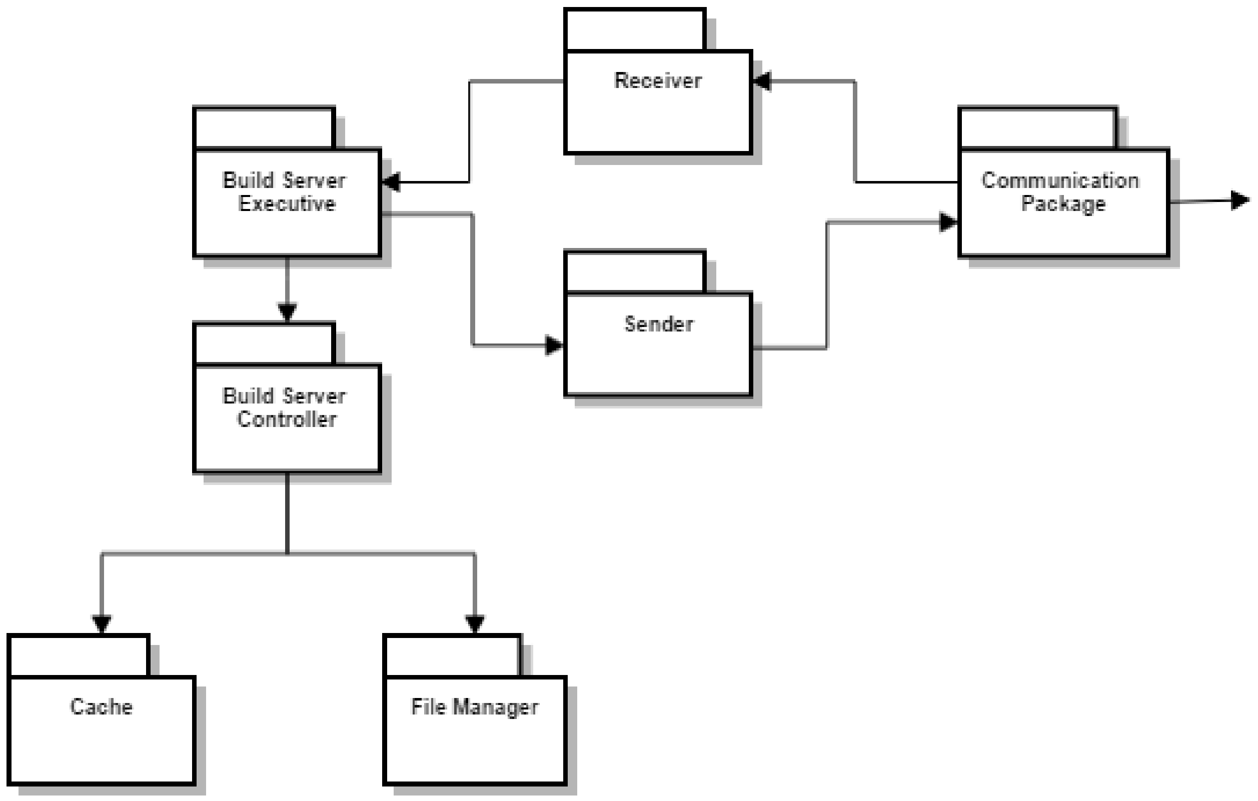


Figure 27 : Package Diagram of Build Server

The Build Server Package Diagram shows the modular Structure of the Build Server subsystem. The Build Side is connected to the Test Harness Server and the Repository Server via communication channel.

Build Server Executive Package

Build Server Executive package receives request from the Test Harness Server to build image of the source code and test suits via communication channel. The Build Server package receives requests from the test harness server via a communication channel. The Build Executive will forward this request to the Build Server Controller for advancement.

Build Server Controller Package

Upon receiving build image request from the Executive package, Build Server Controller package serves Test Harness's request. It checks if the requested code is already in the Cache Memory or not. If it is then it will only request the source code from the Repository Server via communication channel else this package will make request of both source and dependent code to the Repository Server.

Upon receiving requested code from the Repository Server, this package saves entire code into the File Manager package. Controller package then carry out build process in order to make build images. If build gets successful, this package sends build images to the Test Harness Server through communication channel. And if it build image fails it genereates notification for both Repository and Test Harness server.

Cache Package

This package stores the recent successful builds into the Cache. Whenever Test Harness again ask for some build image, Controller Package will look in this package to see that whether it is available here or not. If it's there then controller will utilize it.

File Manager Package

File Manager Package works as a repository of incoming source files, test suits and XML mapping files. This package manages all the files in the build server.

6.4. Build Server's Activity Diagram

The following activity diagram represents the graphical representation of the build server subsystem. It represents the major activities that should happen during the execution Build Server.

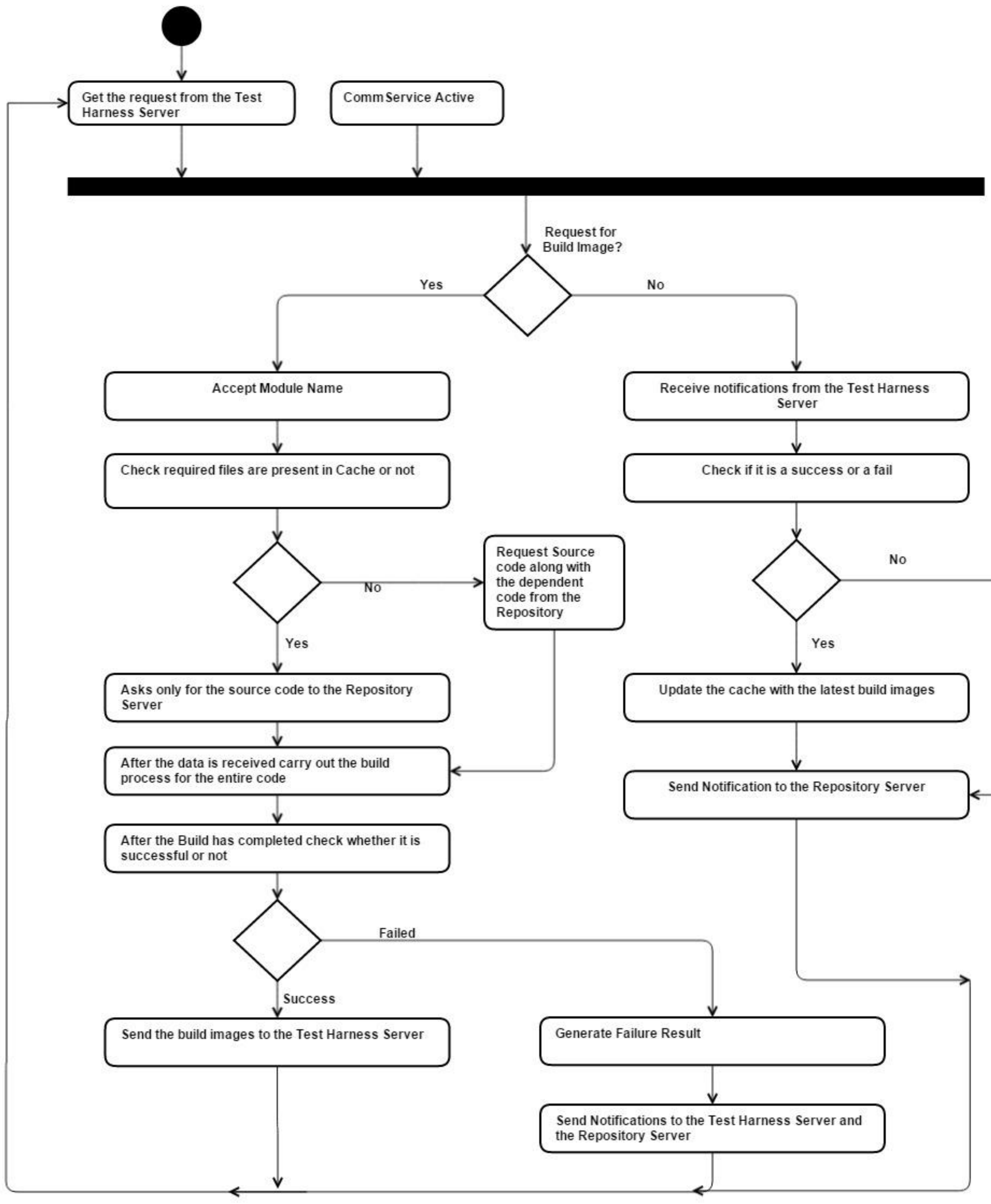

Get the request from the Test Harness Server
Comm Service Active
Request for Build Image?
Yes
No
Accept Module Name
Receive notifications from the Test Harness Server
Check required files are present in Cache or not
Check if it is a success or a fail
No
Request Source code along with the dependent code from the Repository
No
Yes
Yes
Asks only for the source code to the Repository Server
Update the cache with the latest build images
After the data is received carry out the build process for the entire code
Send Notification to the Repository Server
After the Build has completed check whether it is successful or not
Failed
Success
Send the build images to the Test Harness Server
Generate Failure Result
Send Notifications to the Test Harness Server and the Repository Server

Figure 28 : Activity Diagram of Build Server

When the Test Harness Server makes request of building Build Image, following activities will take place at Build Server side:

- Build Server can receive two types of request from the Test Harness server :

  1) For Building build image

  2) Success/Failure Notification

- If request is for building build image then Build Server accepts module name and checks it into the Cache. If it is present there then it only request the source code from the Repository server else it will request both Source and Dependent code from the Repository server.

- It carry out build process on received files from the server and generates build image. If built image is successfully created then it sends it to the Test Harness Server. But if built failed then Built Server notifies the Repository Server and Test Harness Server regarding the same.

- Another request Build Server may receive is success/failure notification from the Test Harness Server. If testing succeed then Built Server stores the previous generated build image into the cache for future use. But if testing gets failed then it sends notification to the Repository to not check-in new module into the baseline.

7. Critical Issues

While designing the architecture of a system, there can be several critical issues which need to be addressed. The lifetime of software is directly proportional to how the system handles these issues. In this section, some of the critical issues related to the CBIS are as follows.

Communication Failure

As being Client Server tool, it might possible that during the file transfer between the Client and Server, connection breaks. In that case the incomplete transfer of files take place at server side. This might lead to the huge problem as data will not be available to the other end.

Solution : To overcome this problem, we must notify Client and Server regarding status of the connection through acknowledgement message. One proposed approach is that : At the end of the file transfer, client always sends dummy file telling server about the termination of the transfer. If server doesn't receive this file until some amount of specified time, then server might thought that there was a incomplete file transfer so it will request again to the client for sending the files again.

File Contention

The CBIS allows multiple clients to access the same server at the same time. The server handles this requests via Queue system. Through this Queue, server provides services by picking each request through child thread. Child threads running concurrently. Now if two or more threads have the request to analyze the same project then this will result in file contention, because these threads will request the analysis of the same files of the project.

Solution : We can introduce locking operation here. If multiple child threads request the same project at the same time, one who gets first access to analyze the project from queue puts up lock so other thread will sleep for some time until the request is served.

Large Check-in Request

User have to face various issues with large file uploads like browser limitations, server configuration issues, timeouts etc.

Solution: Probable solutions for large file uploads can be either to use file chunking or streaming. Chunking should also consist of the ids so that once they are received at the other end they can be combined into a meaningful message.

Huge Check-out Request

Client requires response from server once server analyze a certain request of client. If client specifies the check-out request which contains thousands of files, the final outcome will be huge. This might exceed the capacity of the message size and client might not get all requested modules by server.

Solution: One approach is to transfer requested modules with chunking technique. Out of the many packages, one package will retrieved from the baseline and we will store the results in different chunks according to the size of message. These chunk will passed to the client with sequence number and a global ID which will help client to merge them together.

File Sharing

Consider the case when two processes request the same file for access at the same time. It will throw an exception and this will result in unexpected behavior of the system.

Solution: To overcome from this problem, we can use solution like ConcurrentFileAccess demo provided by Dr. Jim Fawcett which handles all of these exceptions and provides smooth functioning

Load at the Server

CBIS is huge application that deals with the large data sets. It might take long for Operating System to execute it. Because Repository Server has most of the important large modules. It holds baseline where there are too many packages and modules are resides. So the load at the Repository Server will be very much and it needs to be maintained.

Solution: One solution to overcome this problem is to use load balancing algorithms at the server side. Load balancing aims to optimize resource use, maximize throughput, minimize response time, and avoid overload of any one of the resources.

8. Conclusion

This Architecture Report describes the architecture of the Contiguous Build and Integration System (CBIS) that can be used in implementing the system. With keeping in mind the solution of the critical issues, the implementation of this system will not contains the major flaws. In conclusion, CBIS provides following features:

- Provides User Friendly GUI for interaction with the system
- Provides enhancement and modification capabilities to the exiting project
- Provides option to select different functionalities such as Check-in, Check-out, Dependency Analysis, View Reports, Test Entire Baseline.
- Confirms that added module that resides on baseline have no bugs and anyone can analyze it in order enhance it.

We can summarize Continuous Build Integration System as an efficient and robust tool used for continuous build of large applications in the organizations that provides the enhancement and modification capabilities with ease.

9. References

1. Project Helper 2014, provided by Dr. Jim Fawcett
2. Project 5– Quality Assurance Toolset OCD, by John Walthour
3. Project 5 - Quality Assurance Toolset by Arunkumar, Provided by Dr. Jim Fawcett
4. Software : Pencil (for preparing GUI)
5. http://www.gliffy.com : Gliffy, (for preparing Activities and Package Diagrams)