SEPTEMBER 13, 2017

CSE-681 Software Modelling and Analysis

Instructor - Dr. Jim Fawcett

BUILD SERVER

PROJECT#1- OPERATIONAL CONCEPT DOCUMENT

RAMA TEJA REPAKA (SUID#264100460)

Contents

Executive Summary:

Continuous integration of big software systems is one of the most common challenge that we face today in real world. Big software systems contain thousands of packages and perhaps several million lines of code.

Packages needed to be thoroughly tested before moving to the software baseline. If new parts are added to the baseline there may be chances of errors and performance problems so regression testing needs to be done on entire baseline to make sure that things work fine. As in the real-world changes occur frequently through testing of complete software baseline becomes overhead. So best way to intensify this testing practical is to automate the process.

The main aim of the project is to provide continuous integration that is when new code is created for the system we build it we test it in the context of other code which calls it and as soon as the test passes we check in the code and it becomes part of the current baseline. We use several services to efficiently support continuous integration.

We make use of dedicated servers like Mock-Client, Mock-Repository, Build Server, Mock-Test-Harness. Each of them will be having unique responsibilities which are necessary to support continuous integration of software baseline. Each of the dedicated server's responsibilities will be known shortly as we go through the document.

The intended users of this project are Developer, Teaching Assistants and Instructor. The developer will use this document for guidance and implementation in later phases. The TA's and instructor shall use this to check whether all requirements are met all critical issues related to the project are addressed.

Below are some of the critical issues associated with this project and solutions are discussed in detail in further sections.

The Remote Build Server should be easy to use and thus providing a simple easy understandable interface is crucial.

One of issues will be building different types of source code like C++, Java, C # etc. If we use MS Build library then it should be able to build only C# programs but not like Java, python. So, we need to create our own build infrastructure that uses the compilers and tool chains for the target platform

The Build Server may have heavy workloads just before the releases so we should make the building through put for building code as high as is reasonably possible. To handle this critical issue, we may need to implement the process pool

There may be Malformed code which overflow the process stack this should not stop the builder. It should create a new process replacement and reports the build errors to the repository.

There may be also chance of runtime exceptions and client also may crash sometimes

As in the project2 we are not handling multiple requests so it may not be issue but when the scope is extended to project 4 building a communication channel between different pool processes by using WCF may lead to dead-lock situations since we make use of multi-threaded programming to build process pool

The Build Server will be one of the busiest servers in the project. One of the most critical issues is its performance. The time required to complete all the build tasks increases if there are more build requests and more source code files coming from the repository. As a result, there will be overhead on entire system because Mock test harness also takes lot of time to test and prepare reports and send back to the repository and it also should notify the author. This can be partially managed by multitasking and manage of resources

May be due to inconsistency in the Input build request

Demonstrating to the graders that all the requirements have been met without having them to manually enter the data.

Proper versioning control system should exist repository because there may be chances of build failure at such crucial times versioning control helps to resolve needs.

As in the project 2 authentication and authorization may not be issue but when it is extended to project 4 authentication and authorization should be handled when Mock-Client connects remotely to the Repository Contents to do some actions on the code like check- in, check- out etc.

Introduction

CONCEPT AND KEY ARCHITECTURAL IDEAS:

For supporting continuous integration that is when a new code is created for the system, we build and test it in the context of other code which it calls and as soon as the test case passes we check in the code and it becomes part of the current baseline, we make use of

Mock Client which has the interface to submit the code and tests requests to the repository and later it also has the capability to view the results stored in the repository.

Mock Repository which contains all the code documents of the current baseline along with their dependency relationships which are sent to the build server to build test libraries. It also holds the test results sent by the Mock test harness and may also cache build images.

Most Important part that is, Build Server based on the code and build requests from the Mock repository, the build server builds test libraries and submit it to the Mock test harness along with the test requests and send build logs to the repository.

Mock Test Harness based on the test requests and libraries sent from the build server runs tests concurrently for multiple users and it submits results to the repository and notifies the author of the test of the results.

We can demonstrate we are meeting all the requirements to the Graders and TA's by making a logger to console as well. The logger would thus write status to console as well as log file which gives us in sight in to project execution.

REQUIRED FUNCTIONALITIES:

Following are some of the functionalities that will be implemented in the project

1. Build Server should receive all the source code and the test requests sent by the repository

2. Build Server should be capable to build each visual studio project file delivered by the Mock Repository
3. Build Server should report success failure messages of the build attempt
4. Build Server should also display warnings
5. Build Server on success should deliver built library to the Mock Test Harness
6. Mock Test Harness on command shall attempt to load and execute each test library catching any exceptions that may be emitted, and report success or failure and any exception messages, to the Console.

Following are some of the Functionalities that can be added to the Remote Build Server in a real-life scenario.

1. Multiple clients should be able to submit the code and the test requests and then displaying test results accordingly.
2. Clients will be having easy to use graphical user interface to interact with the Repository
3. Build Server should run builds simultaneously for multiple users by using concept of multi - threading.
4. Build server should use the concept of process pooling to handle multiple build requests
5. Build server should use own build infrastructure that is compilers and tool chains for the target platform because it should support various types of source codes like C#, JAVA etc.
6. Client’s authentication should be implemented using secure socket layer to provide security to repository contents
7. Repository should maintain versioning control because it will be helpful in the cases of build failure
8. On command Test Harness should be able to run thousands of tests in a reasonable amount of time
9. Test Harness should also log results separately for multiple users and store them by author name in a repository

ORGANISING PRINCIPLES

Most of the functionalities mentioned above will be achieved by splitting them into smaller tasks Each task, or a group of related tasks, will be put into separate packages that make it easier to understand the operation the overall application. The various packages that make up the application and will include packages that control the program flow, test the various functionalities, process build requests, generate dynamic link libraries and log the results and execution.

USES

The current project mainly focuses on designing and implementing build server and such as its primary uses will be restricted to processing build requests, building libraries, noticing the client, submitting libraries to test harness on successful builds and commanding test harness to execute.

USERS AND IMPACT ON DESIGN

Course Instructor and Teaching Assistants:

These users will focus on testing the implemented build server. This includes checking the various functionalities and evaluating the overall performance of the application.

Impact on Design:

Ideally testing should be designed such that it takes minimum input from the user, and displays the results of the build test effectively in a lucid concise manner.

Keeping this in mind program should include a Test package that will be responsible for demonstrating each of the requirements by running step-by-step through a series of tests. The results should be displayed concisely to facilitate easy comprehension.

Developers:

Developers are the primary users of this Remote Build Server. They work closely with the testers because they should not be wide gap between what developers has produced and what tester needs. They start the coding process and prepare unit test case documents to test their own modules.

Impact on Design:

Ease of use is the major design impact for the Remote Build Server. we need to provide user friendly graphical user interface to the developer along with the remote access facilities.

QA'S

The QA'S need to run several builds to demonstrate that the project meets the requirements. They may build the entire baseline and then later check the logs. As this is kind of automated process it saves lot of time. They also need to have the capability to understand issues and generate test cases. They mainly should have knowledge of execution methods.

Impact on Design:

Performance may be the major concern for them because there may be several build requests. So, build server should be fast enough to process all request and generate libraries.

MANAGERS:

Manager is the main coordinator and work delegator in the system. They make sure that all resources are available to the team and they also prepare overall report.

They make sure that processes are carried out in strict order and check constantly whether meeting deadlines or not.

They also check on going project activities like check test case count, logs etc.

Impact on Design:

Managers need the test results data. Thus, the build server should provide excellent logging facilities with proper time and date stamp.

It should also provide facilities to retrieve graphs showing the amount of load on the build server. These logs should be named, identify the test developer, the code tested including version, and should be time date stamped for each test execution.

POSSIBLE EXTENSIONS:

EXTENDING IT TO PROJECT4

In project 4 we are going to Implement Remote build Server where multiple clients can connect to the repository and check in the code and create multiple builds requests. Multiple requests will be handled by process pool concept. This application, capabilities can be extended by providing a graphical user interface by using WCF on the top of Build Server, mock repository , mock test harness.

LOGGING

Event log processing and analysis play a major role in applications ranging from security management, IT trouble shooting, to user behavior analysis. At the same time, as logs are found to be a valuable information source, log analysis tasks have become more sophisticated demanding both interactive. This application can be extended to support log data stores to support log processing and analysis, exploiting the scalability, reliability, and efficiency commonly found in Remote build servers.

VISUALIZE THREADS

Analyzing application running in complex multithreading environment is difficult because of the behavior of the threads. In this scenario, Remote build Server is much useful to visualize the threads behavior.

Module Structure:

Modern software applications scope is so large so we can include all the functionalities in the single package or single source file. Instead application is developed as a collection of packages. The high-level responsibilities of application are broken down into smaller tasks that must be carried out to implement those functionalities. Logically a similar set of functionalities are grouped together what we call packages. Then various packages interact among themselves to provide fully functional software. The advantages of such approach are easier development, testing and debugging.

The package structures of entire application and Build server is shown in the below figures. The package diagrams are followed by a brief description of each individual package and its interactions with the other packages.

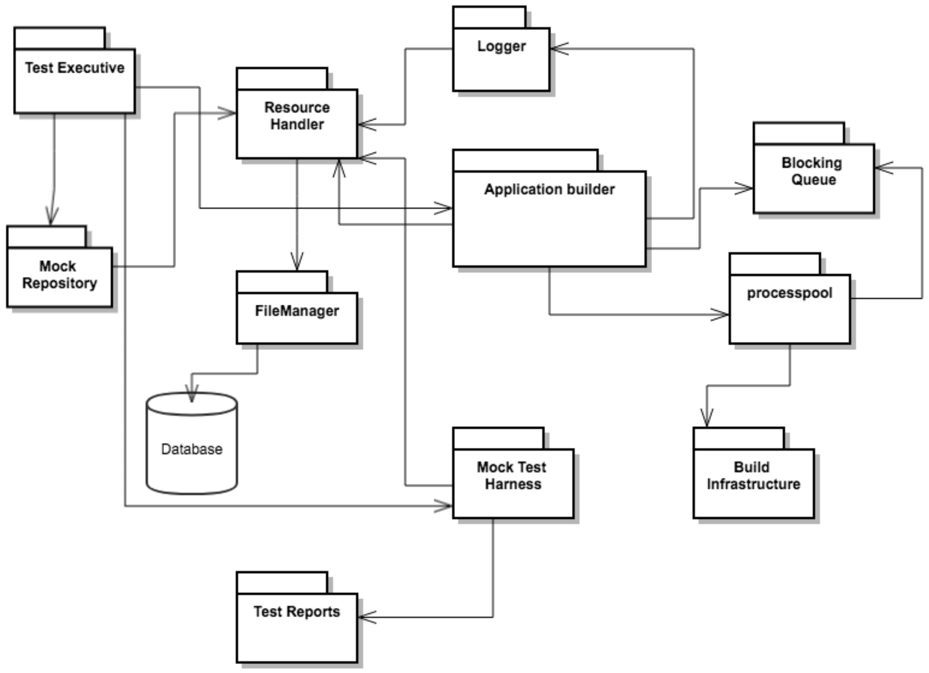


Figure1: Package diagram for Entire Application

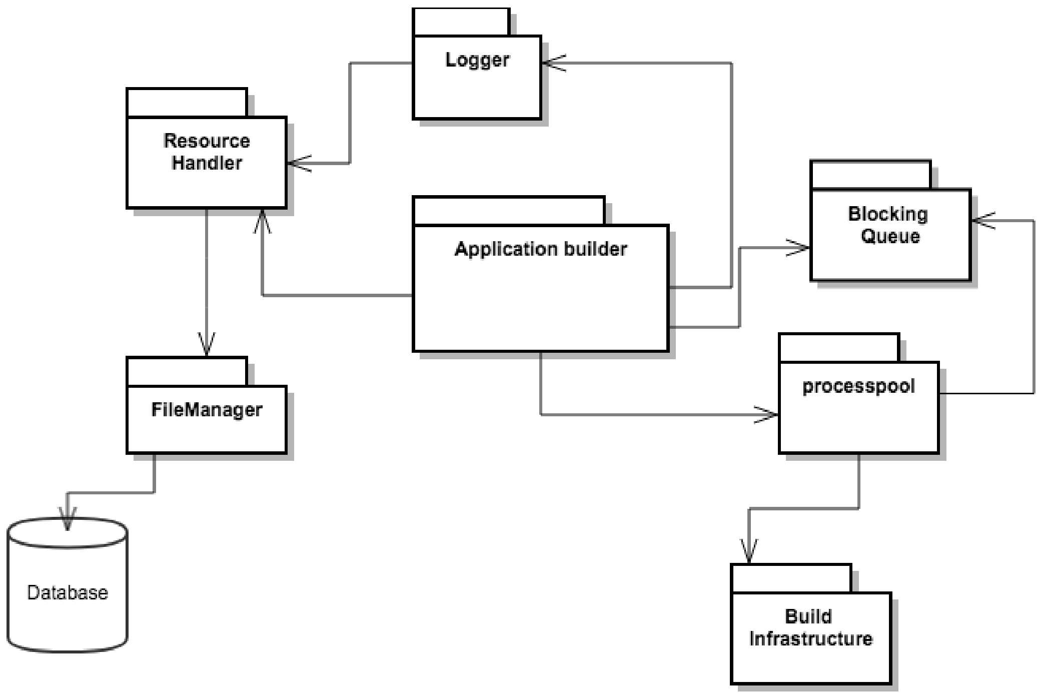


Figure 2: package diagram for Build server

Test Executive:

This is the main package that is responsible for the program control flow because it contains the main method. A series of tests must be performed to demonstrate the Build Server functionalities will be run by the package. It acts as entry point. Test executive directs to various packages to perform a task. It will be responsible for creating below sequence of actions

Mock repository object for sending files and test requests from the repository to a known location to the application builder

Application Builder object for building the source code files, sending library to a location known by mock test harness and logs to repository server

Mock test harness object to load the library received, run tests, generate logs and send to mock repository.

Mock Repository:

This package will be will be responsible for

Making use of database to store and retrieve all the source code files and test requests, sending the source code files and test requests using a resource handler module to the application builder on command, maintaining track of all the build logs send by the application builder and all the test results and logs sent by the mock test harness. It also makes use of logger to record actions performed by mock repository and display in a console.

Resource Handler:

This package will be used by mock repository to copy all the source code files and test request read from the data base to the path that is known to the application builder, application builder to copy all the generated logs back to the mock repository and mock test harness to send all the test reports and logs to the location known by mock repository.

This package interacts with the file manager to locate files from the file system. This package will be used by application builder to receive all the source code files and test requests sent by the mock repository using sender module. This package will be used by mock test harness to send the logs and test results to the mock repository. This package also interacts with the file manager.

File Manager:

This package provides a class, File Manager, and object factory function, create (..). Its mission is to search the directory tree, rooted at a specified path, looking for files that match a set of specified patterns. It provides virtual functions file (…), directory (...), and done(..), which an application can override to provide application specific processing for the events: file found, directory found, and finished processing. The package also provides interface hooks that serve the same purpose but allow multiple receivers for those events.

Logger:

The logger package logs all the activities done by application builder to the file and console. It contains build status, errors and warnings if any. It contains information regarding time stamps of build activities. Application Builder will be using resource handler to send all the log files to a location known to the mock repository.

Application Builder:

This is the main package for entire build process. It uses resource handler package to receive all the files that may be source code files or test requests. If there are multiple build request then insert into blocking queue which is a queue of requests. As in the 2nd project we are not working with multiple request so it may not use the process pool package but in the scope of project 4, it may use the process pool package which contains a set of pooled processes which receives this input request from the queue and then process it , if successful send the library to the mock test harness ,It makes use of build infrastructure package which is responsible for maintaining all the configuration files and tool chains for translating in to those needed by the specific tool chains and finally logger to log the build activity and then resource handler package to send libraries, test requests to the mock test harness and logs to mock repository.

Blocking Queue:

This package is used by the application builder to insert all the build requests into blocking queue and later it will be accessed by process pool package to create build process for each request.

Process pool:

As the project 2 does not require handle multiple build requests simultaneously this package will not be used. But in the scope of project 4 it will be used to reduce the overhead of the application builder by processing multiple build requests simultaneously.

Build Infrastructure:

This package is mainly responsible for maintaining all the configuration files and tool chains of target platforms which will be used while translating builder commands in to those needed by the specific tool chains. This build infrastructure package will be used by the process pool package where each process in the process pool will carry out the corresponding build process in a separate thread.

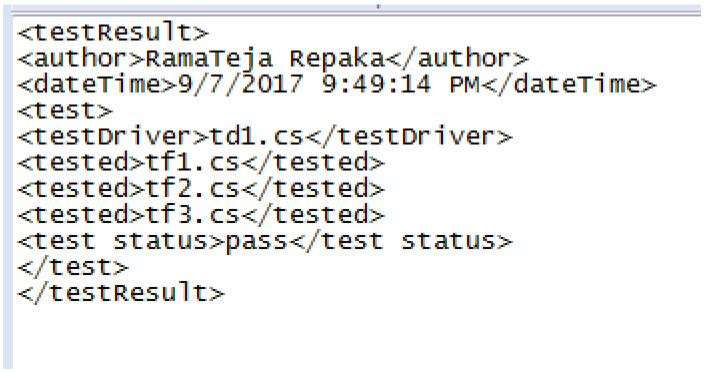
Mock Test Harness:

This package uses receiver package to get all the test requests and then libraries, and later uses loader to load the libraries and blocking queue to maintain series of test requests for each library. It also uses the logger to generate all the logs and then resource handler to sends the logs and test results to the repository server. It uses test reports package to generate test results.

Test Reports:

This package is used by the mock test harness to generate the test results and then send it to the mock repository.

Following is the sample test result that is generated programmatically



```
<testResult>
<author>RamaTeja Repaka</author>
<dateTime>9/7/2017 9:49:14 PM</dateTime>
<test>
<testDriver>td1.cs</testDriver>
<tested>tf1.cs</tested>
<tested>tf2.cs</tested>
<tested>tf3.cs</tested>
<test status>pass</test status>
</test>
</testResult>
```

Data Base contains all the logs sent by the application builder and mock test harness. Data base contents will be accessed through the resource handler package.

KEY APPLICATION ACTIVITIES

The key task of the application is to build the core build server functionality. But during heavy workloads to have high throughput build server make use of process pooling. It means set of processes will be created to handle multiple request where each process will be having the core build server functionality. Build server maintains a queue of build requests and each pooled process retrieves request, process it if successful send libraries to the test harness and retrieve another request.

The overall flow of flow of activities for this project is expressed in this below activity diagram. The core build server and process pooling activities is explained in detail in other activity diagram

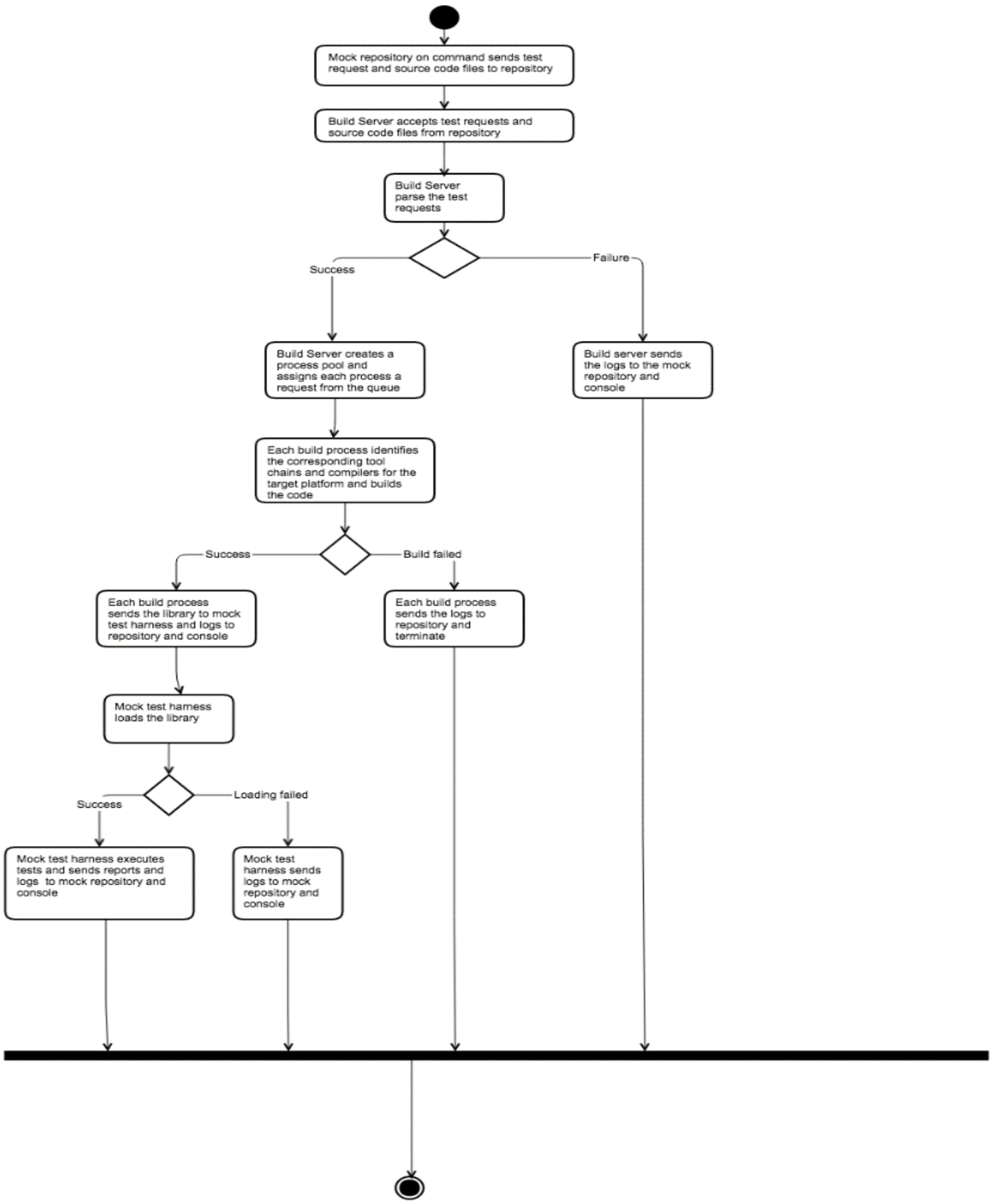

Mock repository on command sends test request and source code files to repository
Build Server accepts test requests and source code files from repository
Build Server parse the test requests
Success
Failure
Build Server creates a process pool and assigns each process a request from the queue
Build server sends the logs to the mock repository and console
Each build process identifies the corresponding tool chains and compilers for the target platform and builds the code
Success
Build failed
Each build process sends the library to mock test harness and logs to repository and console
Each build process sends the logs to repository and terminate
Mock test harness loads the library
Success
Loading failed
Mock test harness executes tests and sends reports and logs to mock repository and console
Mock test harness sends logs to mock repository and console

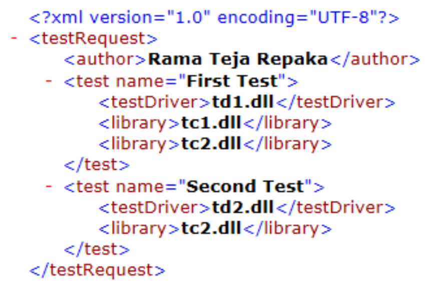
Figure 3: Activity diagram for Entire Application

The main activities involved in the application are described below in a sequence.

Mock Repository sends the input source files, test requests to Build server on command:

Mock Repository acts like a versioning control system where it contains set of files as well as history of changes made to those files which are uploaded to the repository. Mock repository on command sends a set of files along with the test request to the build server. It also stores the logs sent by the build server and mock test harness. In the scope of project 4 this repository functionality can be extended where clients upload all the files and test request through graphical user interface to the repository.

Test request comprises of tests which in turn describes test libraries and test drivers. Below figure represents a sample test request



```
  <?xml version="1.0" encoding="UTF-8"?>
- <testRequest>
     <author>Rama Teja Repaka</author>
   - <test name="First Test">
        <testDriver>td1.dll</testDriver>
        <library>tc1.dll</library>
        <library>tc2.dll</library>
     </test>
   - <test name="Second Test">
        <testDriver>td2.dll</testDriver>
        <library>tc2.dll</library>
     </test>
  </testRequest>
```

Building Source Files

The build server will attempt to build every visual studio project delivered by mock repository. Build process can be done by using MSBUILD library or by creating own build infrastructure. MSBUILD works only for C# projects. Our own infrastructure uses compilers and tools for the target platform.

Potentially each will be having a configuration file for each platform and tool chain that identifies that identifies the path of the tools and it should translate the builder commands in to those needed by the specific tool chains.

In the scope of project 2 I will be using a MSBUILD library which takes a C# visual studio projects an input and tries to build each project and report to the console and the logs regarding the build attempt, errors, warnings, success messages if any.

For other often used languages like C++, JAVA I will maintain separate configuration files and tool chains that identifies the path of C++ and Java tools to translate builder commands into those needed by specific tool chains.

After the build process, it stores all the logs in the mock repository and outputs to the console and handovers all the build library to the mock test harness.

There may be heavy workloads during customer releases so we implement process pooling on the top of these Mother build server. But we are implementing this concept in the scope of project4.

Below activity diagram shows the build process of source code in more concise way.

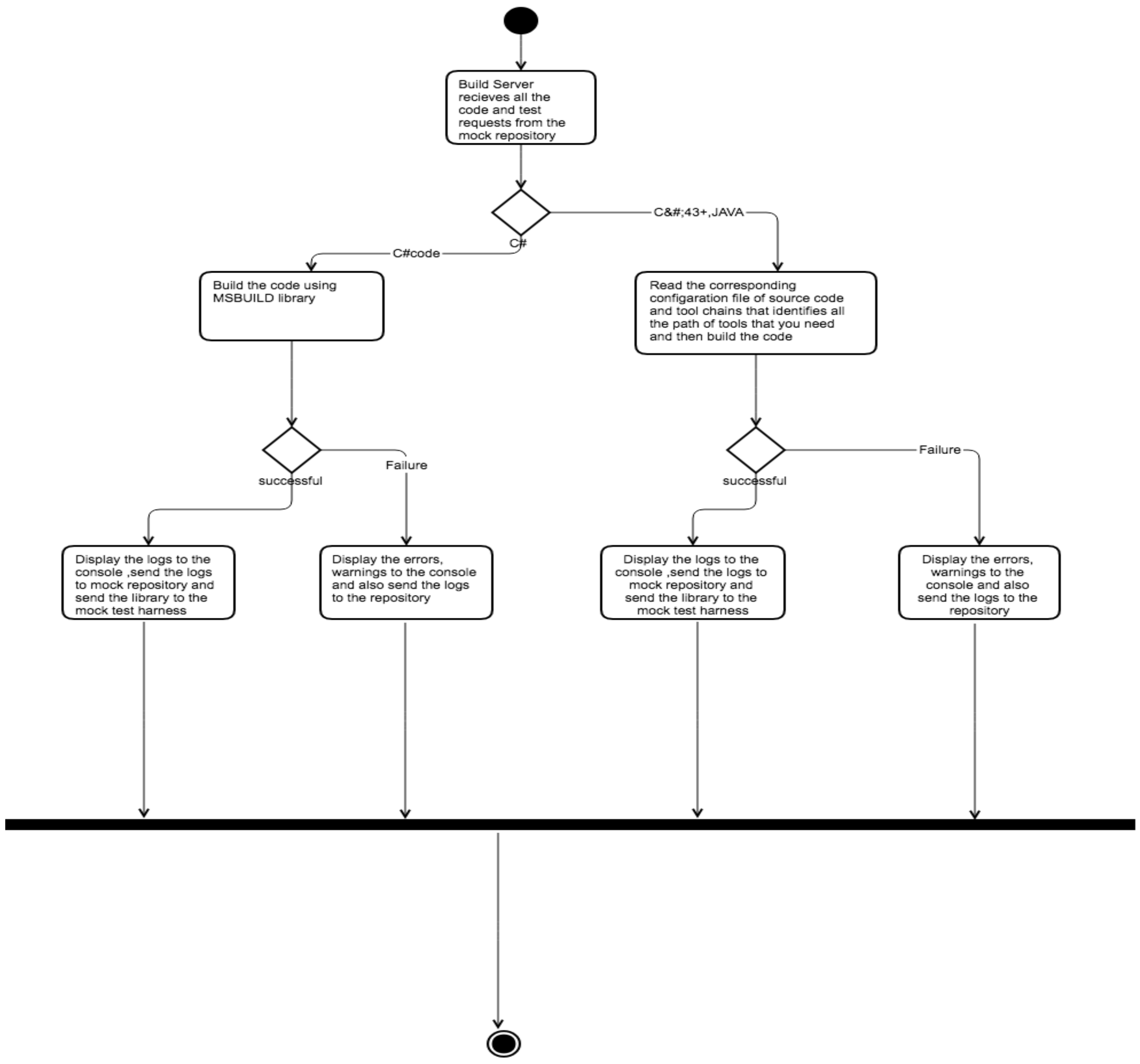

Build Server recieves all the code and test requests from the mock repository
C&#;43+,JAVA
C#
C#code
Build the code using MSBUILD library
Read the corresponding configaration file of source code and tool chains that identifies all the path of tools that you need and then build the code
successful
Failure
successful
Failure
Display the logs to the console ,send the logs to mock repository and send the library to the mock test harness
Display the errors, warnings to the console and also send the logs to the repository
Display the logs to the console ,send the logs to mock repository and send the library to the mock test harness
Display the errors, warnings to the console and also send the logs to the repository

Figure 4: Activity diagram for Build Server

Process pooling

This concept may not be used in the context of project2 but it can be extended in the scope of project4.

Process pool contains set of processes.

These processes will be spawned at startup

Each process retrieves a build request from the blocking queue that build server maintains and then

process retrieves a request

process start processing it

sends the builds log to mock repository

if successful sends the library to the test harness

and retrieve another request

Note that process pools always communicate with the Mother builder processes.

Below activity diagram shows the concept of process pooling in a more concise way

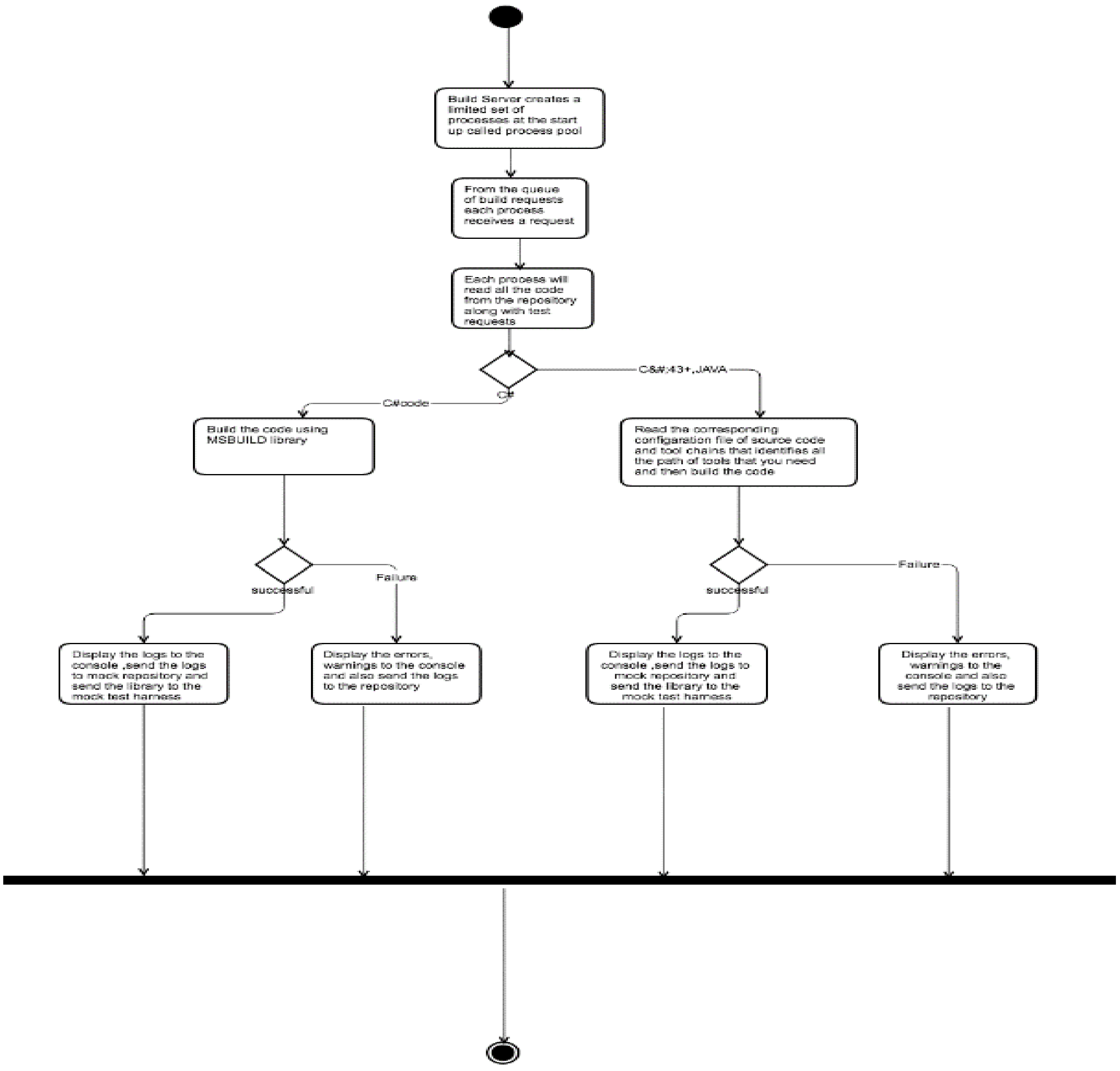

Build Server creates a limited set of processes at the start up called process pool
From the queue of build requests each process receives a request
Each process will read all the code from the repository along with test requests
C&#;43+,JAVA
C#code
C#
Build the code using MSBUILD library
Read the corresponding configuration file of source code and tool chains that identifies all the path of tools that you need and then build the code
successful
Failure
successful
Failure
Display the logs to the console ,send the logs to mock repository and send the library to the mock test harness
Display the errors, warnings to the console and also send the logs to the repository
Display the logs to the console ,send the logs to mock repository and send the library to the mock test harness
Display the errors, warnings to the console and also send the logs to the repository

Figure 5: Activity diagram for process pooling

Delivery of build library to the Mock Test Harness

After the successful build attempt build server sends all the library and the test requests to the Mock Test Harness.

Mock test harness has the following responsibilities

Loading all the library files received from the build server based on test request, starts executing tests based on test requests, logs the results to the console and sends the logs to mock repository. In the scope of project 4 we can extend the functionality to notify the author the tests of results

Display Results

All the requirements shall be demonstrated successfully through a series of discrete tests with display to the console

Critical Issues

Ease of Use

Ease of use will be the one of the major concern for the developers. A complicated user-interface discourages the use of system, thus making a graphical user interface which is easier to understand and simpler to use is crucial.

Solution

Well Structured code design and different use cases should be taken into consideration. Graphical user interface created using windows presentation foundation should be able to upload files and test requests to the repository. But in project2 we are not building graphical user interface users will be most probably given the location of repository as the command line arguments to the automated build process

Building Different Sources

Build Server would be desirable to build sources from several used languages e.g. C, C++, JAVA. If we use our basic build process on MS Build Library then it will not be working for Java and other Linux platforms

Solution

The solution is to create our own build infrastructure that uses compilers and tool chains for the target platform. You need to set up configuration file for each platform, tool chain that identifies the path of the tools and translating your builder commands into those needed by the specific tool chains

Throughput

As the build server is one of the busiest sever it will be having heavy workloads so through put should be as high as possible otherwise build process may take very long time

Solution

The solution is use concept of process pooling. A limited set of processes will be spawned at startup. The build server provides a queue of build requests each pooled process retrieves a request and process it, sends the build log and if build successful libraries to the mock test harness.

Malformed Code

There may be circular set of C++ include statements in the source code   which overflow the process stack which may cause processes to crash

Solution

The idea is to create a new process replacement and report the build error to the repository to achieve this process pools should be always able to communicate with the builder process and each pooled process should have the functionality of build process

Exceptions

Build process may fail if there are any exceptions at run time

Solution

.Net provides to handle exceptions at runtime and developer must try to catch the runtime exceptions

Performance

Build server is one of the busiest server during releases and demos. Therefore, performance of build server is one of the critical issue.

Solution

Multithreading and sharing of resources would decrease a time required to process a build request. Without the use of multithreading each process need to wait in the queue for long time.

Demonstration

Demonstrating that all the requirements are met to the graders and teaching assistants is one of the critical issue

Solution

Printing the outputs to the console for every requirement clearly can demonstrate the requirements are successfully met. Implementing of logger with the date and time stamp across the entire application architecture containing logs of every requirement pass fail status can easily help to demonstrate the requirements are successfully met.

Inconsistency in Input Request

If the input request has some inconsistencies that is it misses some certain fields or values then application might not be able to understand what to do at that situation

Solution

We should provide a validation to the input request to check all the mandatory fields are there in the input request otherwise ask the user to correct the request

Versioning Control

There may be chances of build failures sometimes due to the new code modifications .so entire application servers may be down.

Solution

If proper versioning control is implemented in the repository then it may serve the previous version of working code which can be delivered to make things work instantly.

Authorization

If we use mock repository server which is remotely accessible multiple users may access the application so Authorization is an issue that needs to be addressed

Solution

We must allow only authenticated users to access the mock repository. Multiple levels of access should be developed correspondingly for Developers, Managers, QA's, Instructor etc. Users should be given as much functionality as they are allowed

Conclusion

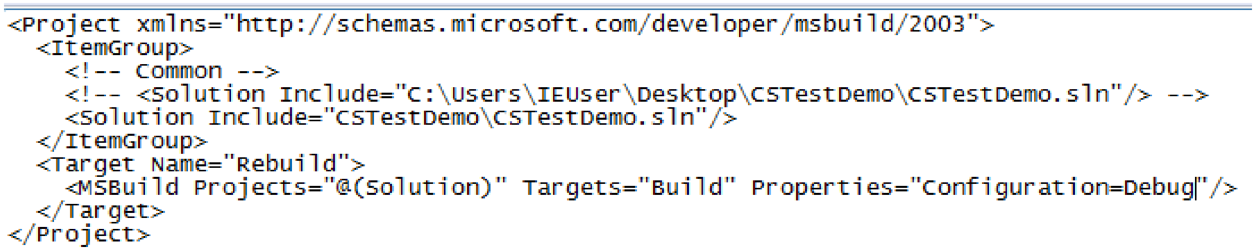
In conclusion to successfully implement big software systems we need to support continuous integration which can be achieved by the federation of servers that is Repository Server, Build Server, Test Harness. Thus, application that we developed provides scalability, flexibility and performance for building big software systems. This OCD explains which actors and how they use the system. We discussed about several use cases. The system is divided into cohesive packages which can interact with each other to perform all the tasks defined in the requirements. It also provides code reusability.

We have covered all the activities being carried out by the system. It describes how events takes place in the system and makes it easy to understand the system. We also discussed few critical issues which can result in serious design flaw if not provided the proper solutions. It can be ensured that from this OCD that system can be developed in considerable period with the available resources.

Prototype:

Visual Studio Project builder code prototype

A prototype has been implemented which takes a Build.proj file as an input which contains the specified paths of the visual studio solution that refers C# packages in sub-directories and attempts to build the project. It also displays success failure or warnings encountered by the build in the console.



```
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <ItemGroup>
    <!-- Common -->
    <!-- <Solution Include="C:\Users\IEUser\Desktop\CSTestDemo\CSTestDemo.sln"/> -->
    <Solution Include="CSTestDemo\CSTestDemo.sln"/>
  </ItemGroup>
  <Target Name="Rebuild">
    <MSBuild Projects="@(Solution)" Targets="Build" Properties="Configuration=Debug"/>
  </Target>
</Project>
```

In the above figure, you can see the specified path of the visual studio solution

provided in the Item Group tag.

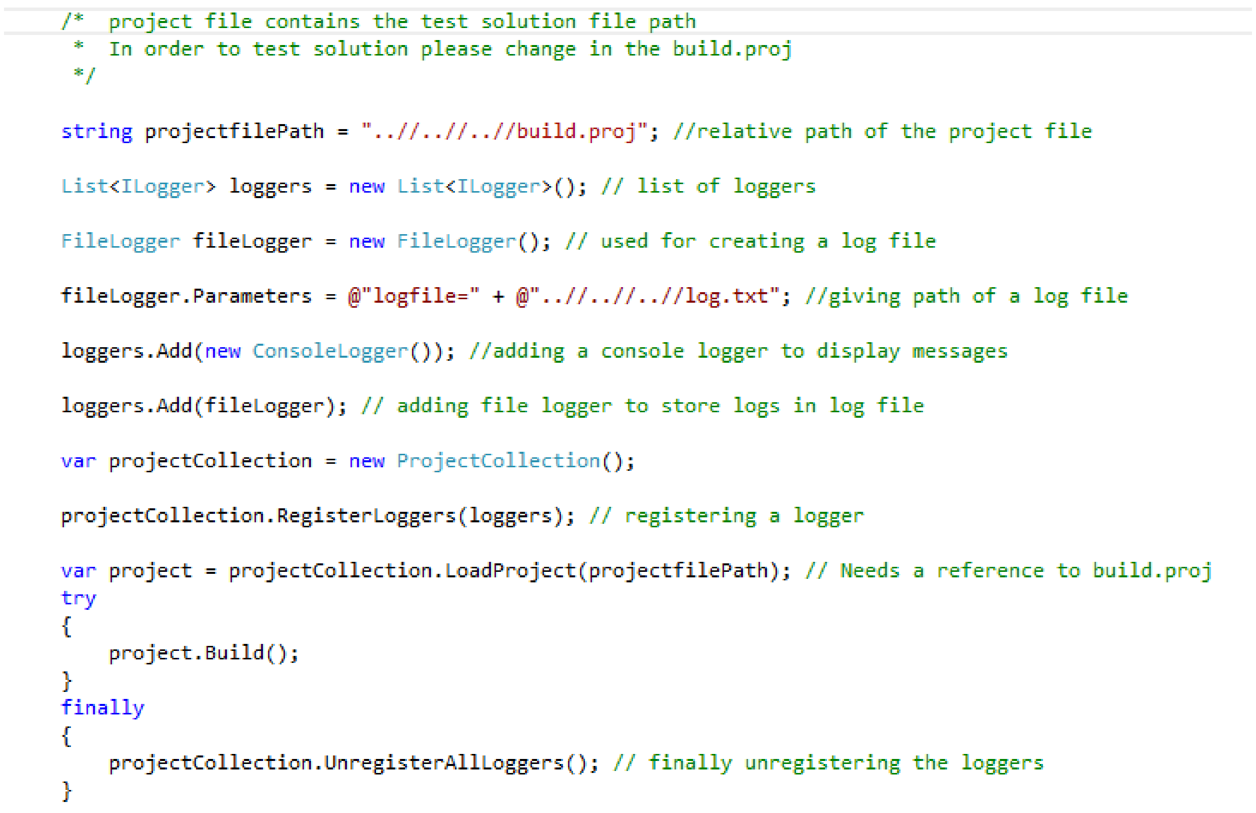
Similarly, if there are more solutions you can add the solution tag with the given solution name in the Item group tag.

You can also change the Target tag name to build and then configuration to debug in the Build.proj file.

By doing this way it gives the flexibility to build more than one solution which satisfies the requirements exactly.

Below figure shows the code snippet which contains the main core logic used to build the

solutions through Build.proj with the comments.

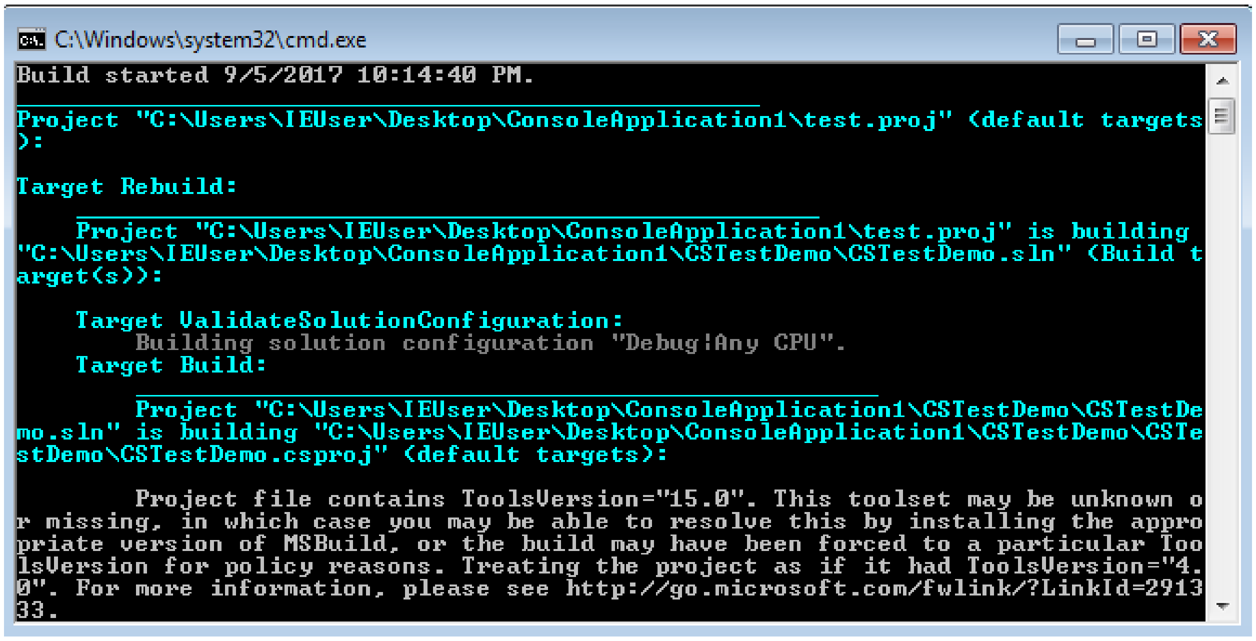


```
/*  project file contains the test solution file path
 *  In order to test solution please change in the build.proj
 */

string projectfilePath = "..//..//..//build.proj"; //relative path of the project file

List<ILogger> loggers = new List<ILogger>(); // list of loggers

FileLogger fileLogger = new FileLogger(); // used for creating a log file

fileLogger.Parameters = @"logfile=" + @"..//..//..//log.txt"; //giving path of a log file

loggers.Add(new ConsoleLogger()); //adding a console logger to display messages

loggers.Add(fileLogger); // adding file logger to store logs in log file

var projectCollection = new ProjectCollection();

projectCollection.RegisterLoggers(loggers); // registering a logger

var project = projectCollection.LoadProject(projectfilePath); // Needs a reference to build.proj
try
{
    project.Build();
}
finally
{
    projectCollection.UnregisterAllLoggers(); // finally unregistering the loggers
}
```

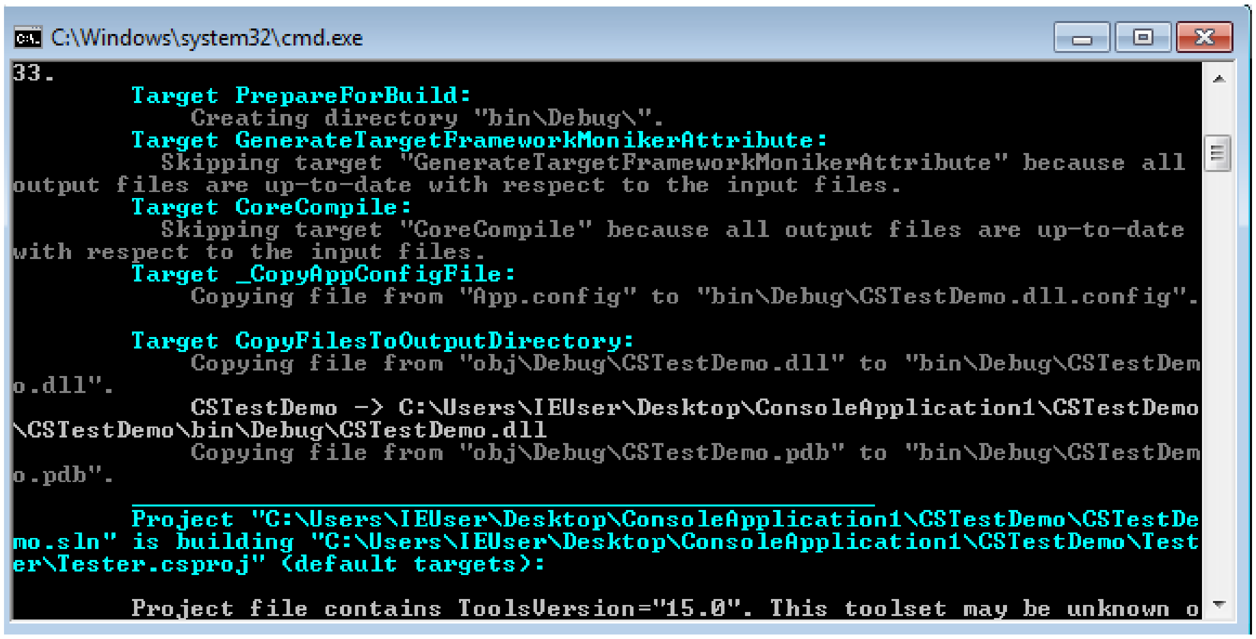
Figure 6: Build project Code Snippet

Output:

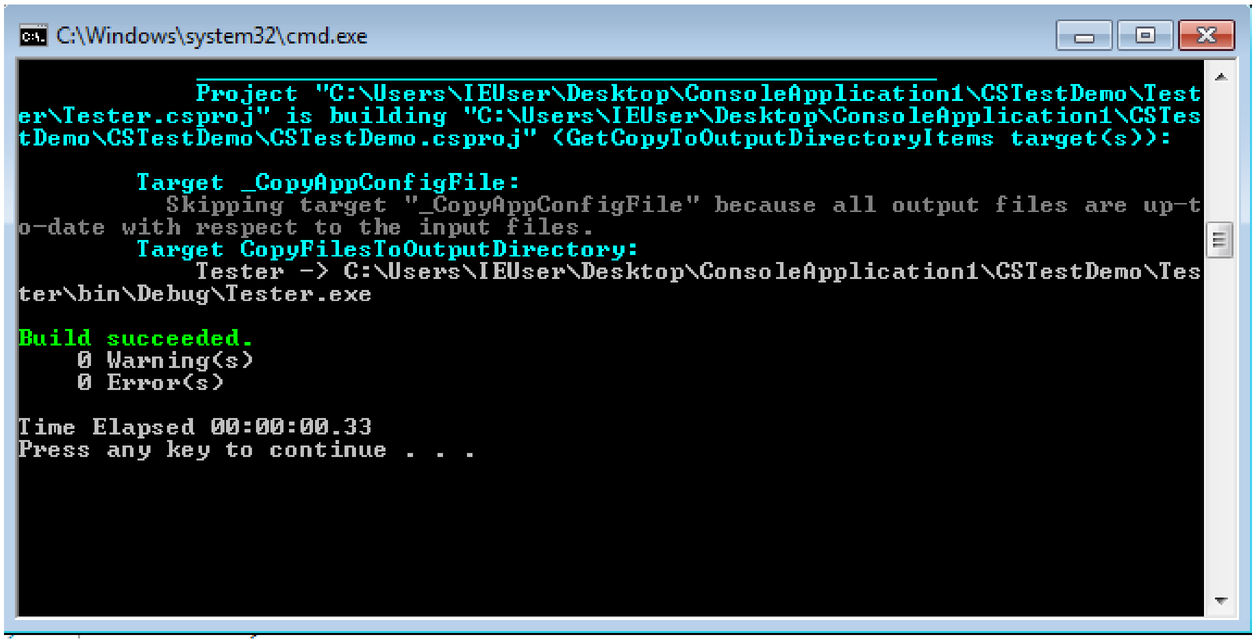
Output is redirected to both console and external log file. Please refer to the below screen shots. It explains the outputs in detail.


C:\Windows\system32\cmd.exe

```
Build started 9/5/2017 10:14:40 PM.
________________________________________________________________________
Project "C:\Users\IEUser\Desktop\ConsoleApplication1\test.proj" (default targets
):

Target Rebuild:
      ____________________________________________________________________
      Project "C:\Users\IEUser\Desktop\ConsoleApplication1\test.proj" is building
"C:\Users\IEUser\Desktop\ConsoleApplication1\CSTestDemo\CSTestDemo.sln" (Build t
arget(s)):

      Target ValidateSolutionConfiguration:
            Building solution configuration "Debug|Any CPU".
      Target Build:
            ________________________________________________________________
            Project "C:\Users\IEUser\Desktop\ConsoleApplication1\CSTestDemo\CSTestDe
mo.sln" is building "C:\Users\IEUser\Desktop\ConsoleApplication1\CSTestDemo\CSTe
stDemo\CSTestDemo.csproj" (default targets):

            Project file contains ToolsVersion="15.0". This toolset may be unknown o
r missing, in which case you may be able to resolve this by installing the appro
priate version of MSBuild, or the build may have been forced to a particular Too
lsVersion for policy reasons. Treating the project as if it had ToolsVersion="4.
0". For more information, please see http://go.microsoft.com/fwlink/?LinkId=2913
33.
```

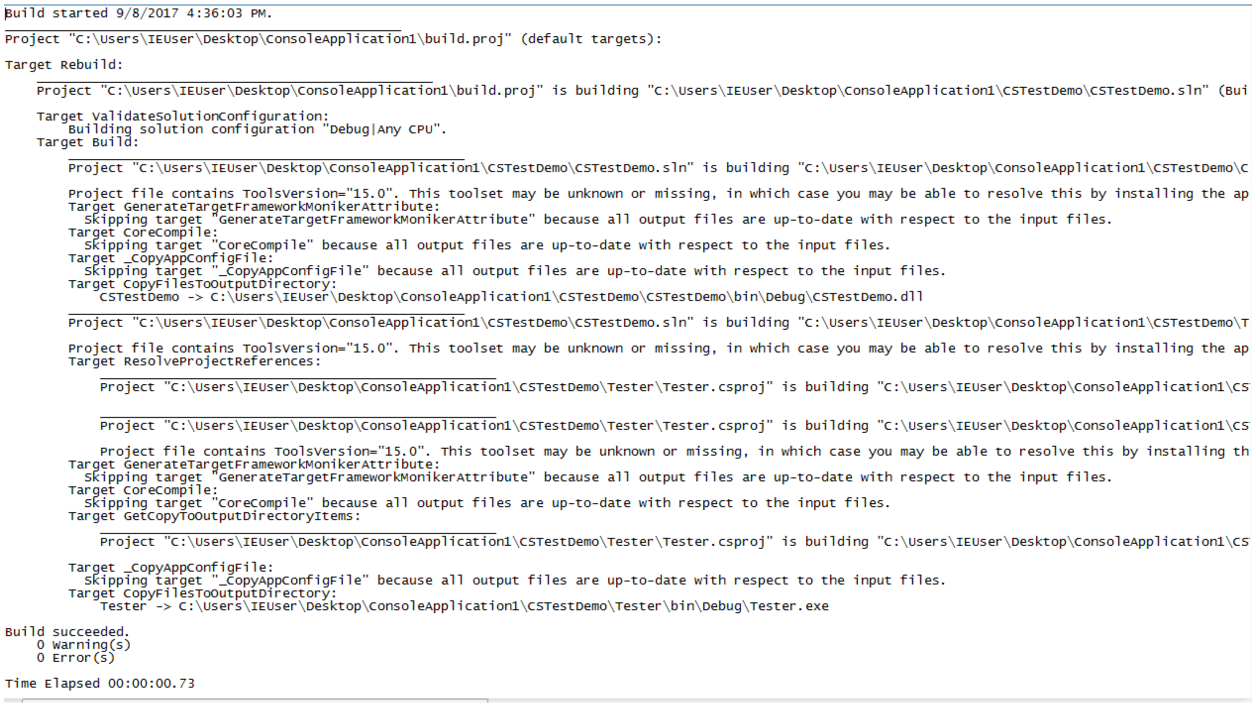

C:\Windows\system32\cmd.exe

```
33.
            Target PrepareForBuild:
                  Creating directory "bin\Debug\".
            Target GenerateTargetFrameworkMonikerAttribute:
               Skipping target "GenerateTargetFrameworkMonikerAttribute" because all
output files are up-to-date with respect to the input files.
            Target CoreCompile:
               Skipping target "CoreCompile" because all output files are up-to-date
with respect to the input files.
            Target _CopyAppConfigFile:
                  Copying file from "App.config" to "bin\Debug\CSTestDemo.dll.config".

            Target CopyFilesToOutputDirectory:
                  Copying file from "obj\Debug\CSTestDemo.dll" to "bin\Debug\CSTestDem
o.dll".
                  CSTestDemo -> C:\Users\IEUser\Desktop\ConsoleApplication1\CSTestDemo
\CSTestDemo\bin\Debug\CSTestDemo.dll
                  Copying file from "obj\Debug\CSTestDemo.pdb" to "bin\Debug\CSTestDem
o.pdb".
            ________________________________________________________________
            Project "C:\Users\IEUser\Desktop\ConsoleApplication1\CSTestDemo\CSTestDe
mo.sln" is building "C:\Users\IEUser\Desktop\ConsoleApplication1\CSTestDemo\Test
er\Tester.csproj" (default targets):

            Project file contains ToolsVersion="15.0". This toolset may be unknown o
```



```
        Project "C:\Users\IEUser\Desktop\ConsoleApplication1\CSTestDemo\Test
er\Tester.csproj" is building "C:\Users\IEUser\Desktop\ConsoleApplication1\CSTes
tDemo\CSTestDemo\CSTestDemo.csproj" (GetCopyToOutputDirectoryItems target(s)):

        Target _CopyAppConfigFile:
          Skipping target "_CopyAppConfigFile" because all output files are up-t
o-date with respect to the input files.
        Target CopyFilesToOutputDirectory:
            Tester -> C:\Users\IEUser\Desktop\ConsoleApplication1\CSTestDemo\Tes
ter\bin\Debug\Tester.exe

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:00.33
Press any key to continue . . .
```

Logs are also stored in the file log.txt, Below screen shots shows the log file contents



```
Build started 9/8/2017 4:36:03 PM.

Project "C:\Users\IEUser\Desktop\ConsoleApplication1\build.proj" (default targets):

Target Rebuild:

    Project "C:\Users\IEUser\Desktop\ConsoleApplication1\build.proj" is building "C:\Users\IEUser\Desktop\ConsoleApplication1\CSTestDemo\CSTestDemo.sln" (Bui

    Target ValidateSolutionConfiguration:
        Building solution configuration "Debug|Any CPU".
    Target Build:

        Project "C:\Users\IEUser\Desktop\ConsoleApplication1\CSTestDemo\CSTestDemo.sln" is building "C:\Users\IEUser\Desktop\ConsoleApplication1\CSTestDemo\C

        Project file contains ToolsVersion="15.0". This toolset may be unknown or missing, in which case you may be able to resolve this by installing the ap
        Target GenerateTargetFrameworkMonikerAttribute:
          Skipping target "GenerateTargetFrameworkMonikerAttribute" because all output files are up-to-date with respect to the input files.
        Target CoreCompile:
          Skipping target "CoreCompile" because all output files are up-to-date with respect to the input files.
        Target _CopyAppConfigFile:
          Skipping target "_CopyAppConfigFile" because all output files are up-to-date with respect to the input files.
        Target CopyFilesToOutputDirectory:
            CSTestDemo -> C:\Users\IEUser\Desktop\ConsoleApplication1\CSTestDemo\CSTestDemo\bin\Debug\CSTestDemo.dll

        Project "C:\Users\IEUser\Desktop\ConsoleApplication1\CSTestDemo\CSTestDemo.sln" is building "C:\Users\IEUser\Desktop\ConsoleApplication1\CSTestDemo\T

        Project file contains ToolsVersion="15.0". This toolset may be unknown or missing, in which case you may be able to resolve this by installing the ap
        Target ResolveProjectReferences:

            Project "C:\Users\IEUser\Desktop\ConsoleApplication1\CSTestDemo\Tester\Tester.csproj" is building "C:\Users\IEUser\Desktop\ConsoleApplication1\CS


            Project "C:\Users\IEUser\Desktop\ConsoleApplication1\CSTestDemo\Tester\Tester.csproj" is building "C:\Users\IEUser\Desktop\ConsoleApplication1\CS

            Project file contains ToolsVersion="15.0". This toolset may be unknown or missing, in which case you may be able to resolve this by installing th
        Target GenerateTargetFrameworkMonikerAttribute:
          Skipping target "GenerateTargetFrameworkMonikerAttribute" because all output files are up-to-date with respect to the input files.
        Target CoreCompile:
          Skipping target "CoreCompile" because all output files are up-to-date with respect to the input files.
        Target GetCopyToOutputDirectoryItems:

            Project "C:\Users\IEUser\Desktop\ConsoleApplication1\CSTestDemo\Tester\Tester.csproj" is building "C:\Users\IEUser\Desktop\ConsoleApplication1\CS

        Target _CopyAppConfigFile:
          Skipping target "_CopyAppConfigFile" because all output files are up-to-date with respect to the input files.
        Target CopyFilesToOutputDirectory:
            Tester -> C:\Users\IEUser\Desktop\ConsoleApplication1\CSTestDemo\Tester\bin\Debug\Tester.exe

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:00.73
```

By the Screenshots it shows that the prototype satisfies the given requirement.

References:

http://www.ecs.syr.edu/faculty/fawcett/handouts/CSE681/Lectures/Project1-F2017.htm

http://www.ecs.syr.edu/faculty/fawcett/handouts/CSE681/Lectures/Project2-F2017.htm

https://stackoverflow.com/questions/6511380/how-do-i-build-a-solution-programmatically-in-c

http://www.ecs.syr.edu/faculty/fawcett/handouts/CSE681/Lectures/StudyGuideOCD.htm