The Federation: Build Server, Repository, and Test Harness

Operational Concept Document
Christina Tobias
Syracuse University
CSE 681

September 13, 2017

Table of Contents

TABLE OF CONTENTS 2

1. EXECUTIVE SUMMARY 4

2. INTRODUCTION 4

3. USES 6

3.1 STUDENT 6
3.2 TEACHING ASSISTANTS AND PROFESSORS 7
3.3 COMPANIES 7
3.4 THE REPOSITORY AND TEST HARNESS SERVERS 7
3.5 DEVELOPERS 8
3.6 QUALITY ASSURANCE ENGINEERS 8

4. APPLICATION ACTIVITIES 8

4.1 REPOSITORY 8
4.1.1. RECEIVE INPUT FROM CLIENT 8
4.1.2 STORE CODE AND LOG FILES 8
4.1.3 DEPENDENCY INFORMATION 8
4.1.4 STORES LOGS AND CACHES BUILD IMAGES 8
4.1.5 CREATE TEST REQUESTS 9
4.1.6 SENDS TEST REQUESTS AND CODE TO BUILD SERVER 9
4.1.7 ADDITIONAL FUNCTIONALITIES 9
4.2 BUILD SERVER 9
4.2.1 RECEIVE CODE AND TEST REQUESTS FROM REPOSITORY 10
4.2.1 PARSE XML FILES 10
4.2.1 BUILD CODE 10
4.2.2 SEND TEST REQUESTS AND CODE TO THE TEST HARNESS 10
4.2.3 SEND BUILD LOGS TO THE REPOSITORY 10
4.3 TEST HARNESS 10
4.3.1 RECEIVES TEST REQUESTS AND CODE FROM THE BUILD SERVER 10
4.3.2 RUNS TESTS 10
4.3.3 SENDS LOG FILES TO THE REPOSITORY 10
4.4 CLIENT 10
4.4.1 RECEIVES AND SENDS INFORMATION TO THE REPOSITORY 11
4.4.2 RECEIVES NOTIFICATIONS FROM THE BUILD SERVER 11
4.4.3 RECEIVES NOTIFICATIONS FROM THE TEST HARNESS 11
4.4.4 DISPLAYS INFORMATION TO USER AND ALLOWS USERS TO SELECT/BROWSE FILES 11

5. PARTITIONS 11

5.1 CLIENTDISPLAY 12

5.2 CLIENTREPOCOMM 12
5.3 CLIENTBUILDCOMM 13
5.4 CLIENTTESTCOMM 13
5.5 REPORECEIVECODE 13
5.6 REPOSTORE 13
5.7 REPOCREATETESTFILES 13
5.8 REPOSEND 13
5.9 REPODEPENDENCY 14
5.10 BUILDSERVERRECEIVE 14
5.11 XMLPARSE 14
5.12 BUILDSERVERBUILD 14
5.13 BUILDSERVERSEND 14
5.14 BUILDSERVERSENDREQUESTS 14
5.15 TESTRECEIVE 14
5.16 TESTRUN 15
5.17 TESTSEND 15
NOTE: 15

6. CRITICAL ISSUES 15

6.1 FAILED BUILDS 15
6.2 VERSIONING 15
6.3 USER FRIENDLINESS 16
6.4 FAILED COMMUNICATION 16
6.5 AUTHENTICATION 16
6.6 TIME 16
6.7 TESTS FAIL OR SUCCEED 16
6.8 PERFORMANCE 17

7. PROTOTYPING 17

1. Executive Summary

In industry, build servers are created by companies to support their individual project architecture. Each company produces their own build servers to be able to customize the way their code is built and allows for them to be able to add additional functionalities to their own federation, which will usually consist of at least a repository and test harness.

The users for this project's federation include the student, teaching assistants and professors, companies, the repository and test servers, developers, and quality assurance engineers. Understanding what each user needs out of the federation is essential to building a good product.

The federation is built around four main components, the client, which includes the GUI, the repository, which holds the code, the build server, which builds the code, and the test harness, which runs the built tests. Every partition for each component is separated according to their main functionalities or communication between other components.

The critical issues include builds that fail, versioning, user friendliness, failed communication, authentication, time constraints, and the success or failure of tests. These are discussed in section 6.

2. Introduction

Build servers are necessary for all developers in order to build and eventually run their code. Companies and individual developers can use personalized build servers to easily maintain their code. Build servers are necessary because they can run automatically, tell developers or teams that their code is broken, and simplify the check-in and build process for developers.

The purpose of this project is to implement a build server that works with a mock repository and test harness to allow users to submit code that will be tested. The federation will:

- Provide users with a GUI that allows them to interact with the repository, build server, and test harness and see pertinent information regarding the build and test processes.
- Allow users to upload code.
- Build the code.
- Run the provided tests given in the code.

The federation will consist of a mock repository, build server, and test harness that communicate with each other to build and test code. Users will be able to choose files and tests that they want to build and run. They will be able to see information, including errors, pertinent to the build and test processes.

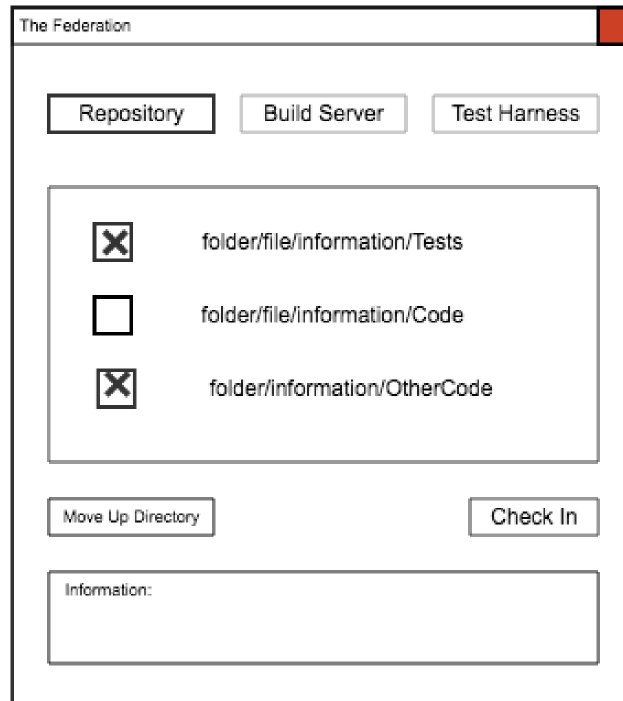

Figure 1: A sample GUI

The mock GUI above in Figure 1 shows what the repository GUI might look like. There will need to be information about the files they can upload, the dependencies, the tests that can be run, and information from the repository. Under the build server tab, there will be information on the state of the build server, if it is ready, building, built successfully, or failed at building the code. The test harness tab will show if the tests failed, succeeded, or have yet to be run.

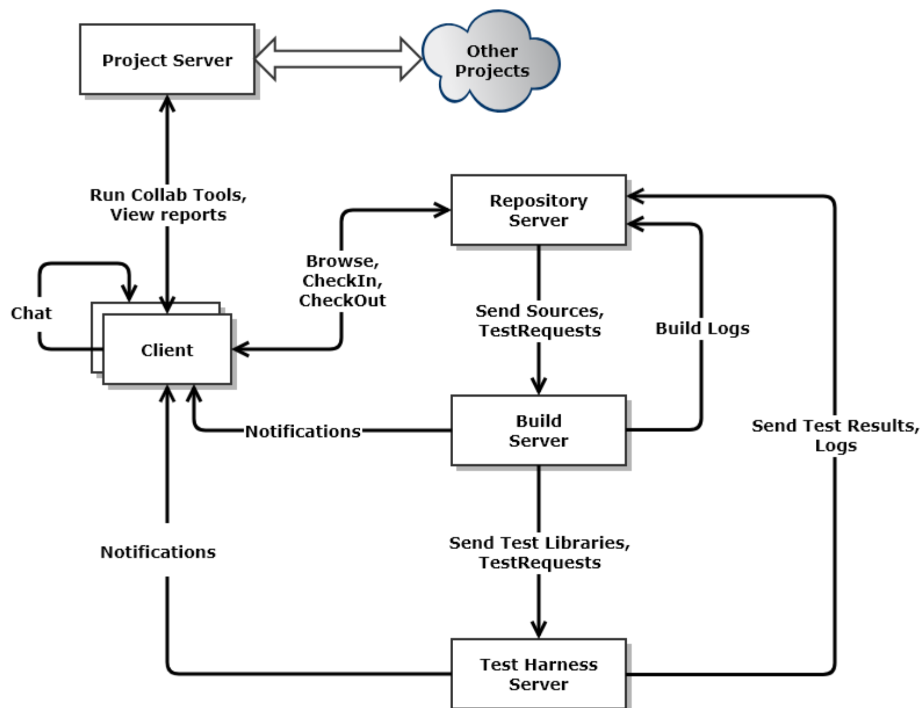

Figure 2: Overview of the Federation
(Taken from the project statement)

The architecture of the design is developed around the four main components of the project: the client, the repository, the build server, and the test harness. Each of these components is broken down into packages that define one large tasks or similar smaller tasks. The end product will have each of these components as separate processes or threads, so planning them as individual components will help as the project is built.

3. Uses

The users of the build server are the student, teaching assistants and professors, companies, the repository and test servers, developers, and quality assurance engineers. Focusing on the needs of each will make the federation work better for all users.

3.1 Student

Each student who builds the federation will need to show the teaching assistants and professor that their code works correctly. She will show that each task is implemented and functions as it should. She will use the federation as a learning tool to

understanding more about how to design larger systems, how C# works, and how to build separate servers that can work together.

For the student, it is essential that the code works well and that there is error checking for any problems that might occur while the code runs.

3.2 Teaching Assistants and Professors

The teaching assistants and professor for this course will use this project as a test to see whether the student understands the material taught in the class. They will have the student show what has been implemented and how the federation works in their implementation. They will use it to determine a grade for the student.

The teaching assistants and professors need a user friendly GUI to be able to choose code to submit and tests to run. They will need to able to see if the tests failed or succeeded easily.

3.3 Companies

Although companies are not going to use this implementation of the federation, a fully functioning federation could be used by a company to build and test their code. It is vital for a company to have a build server that functions in the appropriate way for their system architecture.

A company would need to have a reliable build server that is secure and can be used by many different employees. They would require for it to be easy to maintain and use. They would need only pertinent information to be shared on the GUI, but more detailed information easy to access (i.e. log files).

3.4 The Repository and Test Harness Servers

The repository and test harness servers will directly use the build server through communication. The repository will send code and test requests to the build server. Once the code is built, the test harness server will receive communication that tests are ready to run. The built code will be sent to the test harness and the tests will run.

The repository and test harness need to have simple, yet efficient communication with the build server. The test harness must understand what to do when the build server talks to it, which would be to run the tests.

3.5 Developers

Although the student is the developer for this project, developers at companies would use the federation to submit and test their code.

It would be vital for them to be able to easily submit their code and tests, and receive information back that allows them to see the status of their code.

3.6 Quality Assurance Engineers

Quality assurance engineers are vital to the health of any set of code. They will write tests and look at the output from the build server and test harnesses to ensure the code is functioning correctly.

The GUI would need to provide them with vital information to the health of their team's code. For this project, the student is both the developer and the quality assurance engineer.

4. Application Activities

4.1 Repository

The repository is vital for storing files and communicating with the client and build server.

4.1.1. Receive Input from Client

The repository will receive file names and test information from the client to later be stored. The file names will be complete paths.

4.1.2 Store Code and Log Files

The repository will store the files and test requests after it receives the appropriate information from the client. It will save them in a specialized folder that it creates, which will copy the files from the original source to its own folder.

4.1.3 Dependency Information

The repository will be responsible for knowing the dependencies of each file, so that the build server can build appropriately. It will run an algorithm that will check the dependencies for each file. It will place this information into an XML file to later be sent to the build server.

4.1.4 Stores Logs and Caches Build Images

The repository will store the build logs and test harness logs. In a full product, it would also cache build images, which is a second priority in this project.

4.1.5 Create Test Requests

The repository will build test requests as XML files that the build server can later parse. It will take input from the client to build these XML files.

4.1.6 Sends Test Requests and Code to Build Server

The repository will send test requests, code, and build files (i.e. dependency information) to the build server using WCF.

4.1.7 Additional Functionalities

If the repository were going into production, versioning, multiple users, pulling, and merging would be necessary. With versioning, a user could see past code they submitted and past builds. Multiple users would be able to use the same repository by allowing more branches to be made that can later merge. Pulling would allow multiple users to also make local changes to the same repository and push it back. Merging would look at the differences between the code that is current and the code that is being pushed to ensure the code that changes is correct.

4.2 Build Server

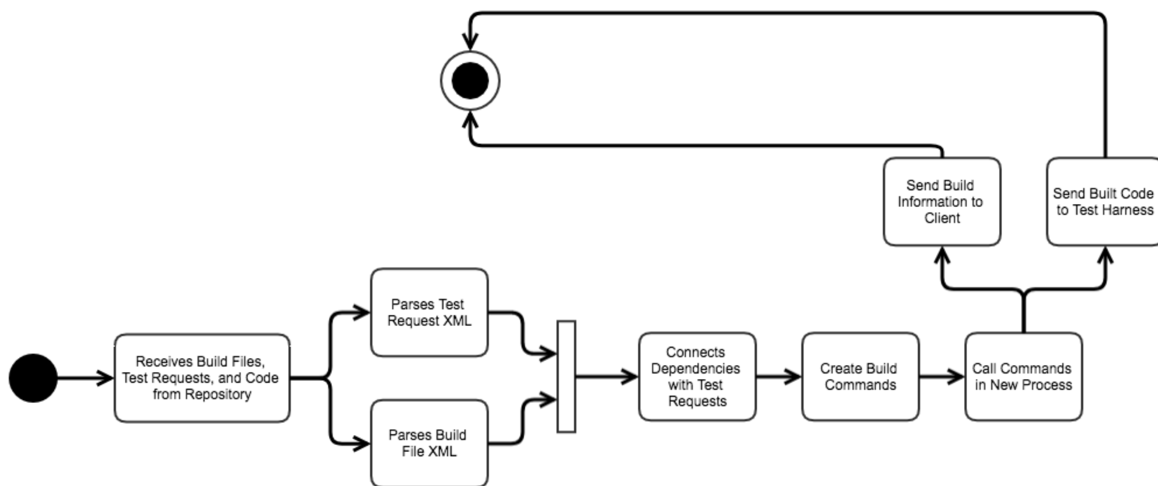

Figure 3: Activity Diagram for Build Server

The build server is the most important part of this federation. It will work by receiving files from the repository, parsing the XML files, connecting the dependencies, creating the build commands, and calling the commands. The information and data from the build server will then be sent to the client or test harness. Below there is more detail about how the build server will work.

4.2.1 Receive Code and Test Requests from Repository

The repository will send code, test requests, and build files to the build server, which the build server will receive using WCF.

4.2.1 Parse XML Files

The build server will be responsible for parsing both the build file and test requests. The build files will hold the dependency information so that the files that need to be built together will be. The test requests will tell the build server which tests need to be built.

4.2.1 Build Code

Once the XML is parsed, the information will be used to connect the dependencies with the test requests. This will combine the information to create a list of build commands that the command line can understand. Processes will run, probably executing cmd.exe, allowing the build commands to run according to their language. Each test will have its own build command, which will include all the files it depends on.

4.2.2 Send Test Requests and Code to the Test Harness

Once the builds are complete, the built code will be sent to the test harness using WCF.

4.2.3 Send Build Logs to the Repository

The build logs will be sent to the repository for the users to be able to access through the client. It will also send details to the client about the status of the build server. The communication will be using WCF.

4.3 Test Harness

The test harness will run the tests that the build server sends it.

4.3.1 Receives Test Requests and Code from the Build Server

The test harness will receive the built code from the build server using WCF.

4.3.2 Runs Tests

Once the built code is received, the test harness will run the tests. Details will be logged in a file.

4.3.3 Sends Log Files to the Repository

The resulting test log from running the tests will be sent to the repository where it will be accessible from the client for users. Information about tests passing or if at least one failed will be sent back to the client using WCF.

4.4 Client

The client’s main responsibilities is to give and get information to and from the user.

4.4.1 Receives and Sends Information to the Repository

The client is responsible for receiving and sending information and data to and from the repository using WCF.

4.4.2 Receives Notifications from the Build Server

The client will receive notifications from the build server. This will be minimal information regarding the state of the build server. The client will receive the state the build server is in, which will be ready, building, success, or failure depending on where in the build process it is.

4.4.3 Receives Notifications from the Test Harness

The client will also receive information from the test harness regarding the success or failure of the tests. It will be told if all the tests passed or if at least one failed.

4.4.4 Displays Information to User and Allows Users to Select/Browse Files

The client will be responsible for the GUI. It must display information to the user that the user can understand and will allow the user to choose the files to build and tests to begin. There will be a tab for the repository, build server, and test harness allowing the information to be organized well.

5. Partitions

The partition diagram below shows how each partition works with the other and the flow of one partition to another. Each partition is explained below.

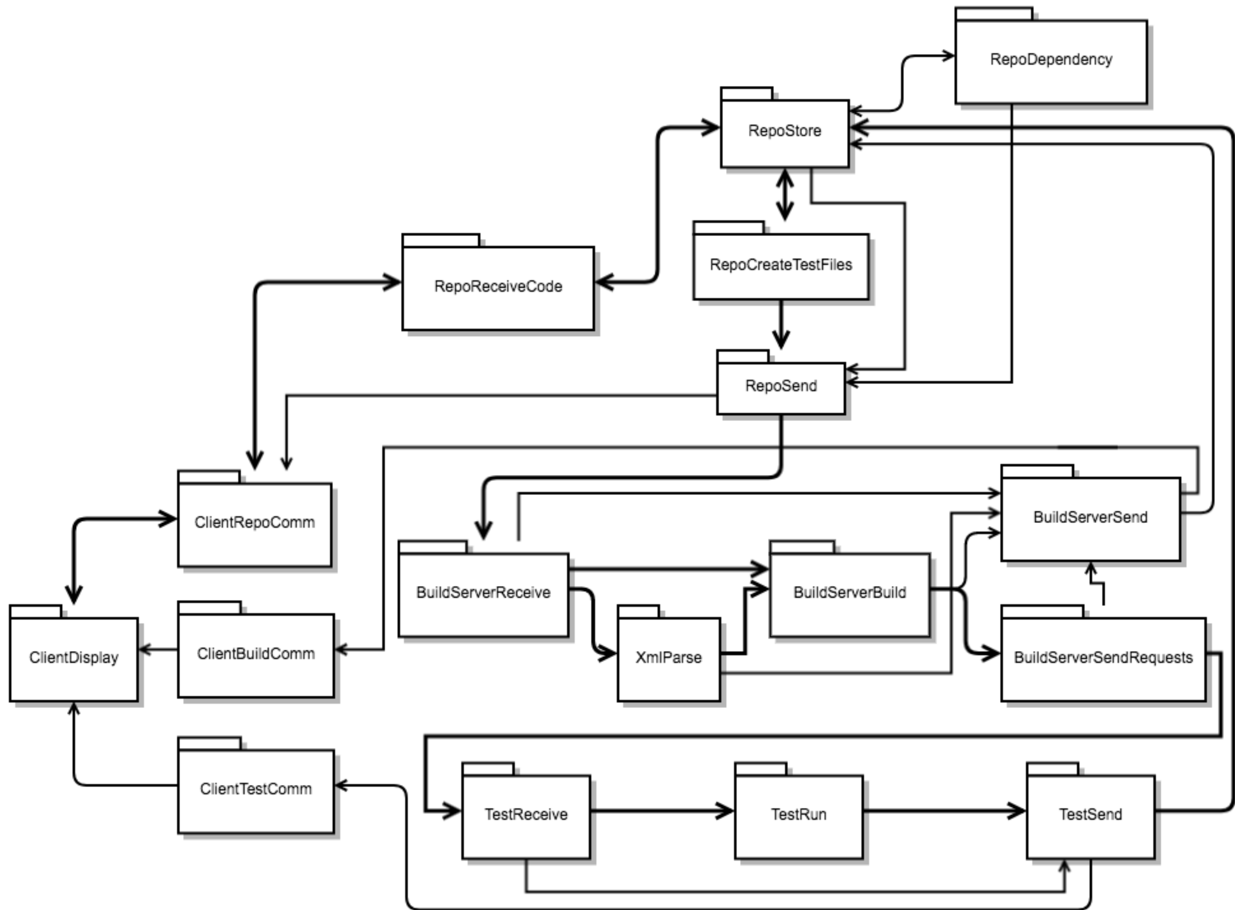

Figure 4: Partition Diagram for the federation

5.1 ClientDisplay

This will be the GUI, showing all the information received from the repository, build server, and test harness. It will also display file and test information to be sent to the repository. It will work with the ClientRepoComm, ClientBuildComm, and ClientTestComm to be able to show users the information coming from each part of the federation.

5.2 ClientRepoComm

This will receive and send all communication to and from the repository. It is responsible for sending file and test information, including the log files that are in the repository, and receiving data about the success/failure of the repository's functionalities. This will be implemented using WCF.

5.3 ClientBuildComm

ClientBuildComm is responsible for the communication between the client and the build server. The state of the build server will be sent from the build server to the client using WCF.

5.4 ClientTestComm

This will send information from the test harness to the client regarding the success/failure of the tests run. It will send either success if all tests succeed or failure if at least one test fails.

5.5 RepoReceiveCode

The repository will receive code and test information from the client. This will also be the interface that sends information back to the client from the repository. This information includes the log files from the build server and test harness and errors regarding the repository.

5.6 RepoStore

The repository will store all the files that are necessary to build the code, including information about which tests to run. This will also store the logs from the build server and test harness. If there were more time in this project, this would also store cached builds that would be sent back from the build server. RepoStore is also responsible for creating the build files for the build server.

5.7 RepoCreateTestFiles

This will create the test requests and build files that will later be sent to the build server. The test requests will be XML files that contain information about the tests that need to be run. The build files will also be XML files that hold dependency information. This is essential to ensure the code can be built.

5.8 RepoSend

This will send the code and test requests to the build server. It will also send any pertinent information to the client.

5.9 RepoDependency

This will find the dependencies in the code and will store this information for RepoStore to use when creating the appropriate build files. This data will be placed in an XML file.

5.10 BuildServerReceive

BuildServerReceive will receive the code, build files, and test requests from the repository. This will contain the information needed for the build server to build the code appropriately.

5.11 XmlParse

XmlParse will parse the XML test requests for the build server. The build server will make a list of the tests that need to be built. If the file that a test is in does not exist, it will relay that information back to the client using WCF.

5.12 BuildServerBuild

This will build the code using the code and build files from BuildServerReceive. This will be done by spawning a process to build C, C#, and C++ files separately. It will use the command line commands and will go through each language. The dependency information will be known, so the files that depend on another will be built together.

5.13 BuildServerSend

Critical information and the log files from the build server will be sent through BuildServerSend to ClientBuildComm and RepoStore, respectively.

5.14 BuildServerSendRequests

The built code from BuildServerBuild will be sent to the test harness by way of TestReceive through WCF.

5.15 TestReceive

TestReceive will get the built code from BuildServerSendRequests. It will then send the build code to TestRun.

5.16 TestRun

TestRun will run the code. If a test fails, it will use TestSend to inform the user. If all succeed, TestSend will also inform the user of the success.

5.17 TestSend

This will send the log files to the repository and send success or failure information to the user.

Note:

Each partition should connect to their main component's (repository, build server, or test harness) send partition. This will ensure if any partition has an issue, the user can be aware of the problem.

6. Critical Issues

6.1 Failed Builds

When builds fail, no tests can run. The user also does not know that the builds failed unless they are told.

Solution: If a build fails, information will be sent to the user and displayed in the GUI. The log file will also be sent to the repository for the user to look up more detailed information regarding the error.

6.2 Versioning

When files are uploaded to the repository, they will replace the existing files and the old files are lost.

Solution: Although this will not be implemented within this project, versioning would be expected of a production ready repository. With versioning, every time code is uploaded to the repository, it would be placed in a time stamped folder that would become the "current" folder to work out of. This would allow users to see past versions of code in case they need it again.

6.3 User Friendliness

A lot of information will be passed around the federation, which might overwhelm a user. The GUI could be cluttered and hard to use.

Solution: The GUI will be simple and clean, only displaying information that is pertinent to the user to understand what is happening with the repository, build server, or test harness. All detailed information will be in the logs from the build server or test harness that the user can access from the repository. It is critical that the GUI have enough information for the user, but not too much.

6.4 Failed Communication

If the server goes down or connection is lost from one process to another, the federation will not function and everything will stop working together.

Solution: There will be error handling regarding failed communication, which will display to the user that connection failed. If it fails, it will continuously try to reconnect until the process dies or there is connection.

6.5 Authentication

If an unauthenticated user tries to upload to the repository, they could overwrite existing code.

Solution: This will not be addressed in this project, but in a production ready federation, there would need to be certificates ensuring the correct users have correct permissions to upload code.

6.6 Time

An important issue is that there is a limited amount of time to implement this project.

Solution: In order to fully implement what is required, careful planning has been taken and some parts have been scoped out of the project.

6.7 Tests Fail or Succeed

If tests fail or succeed, the user will not know since the test harness is it's own process that does not itself have a GUI.

Solution: The test harness will communicate with the client to share information about the success or failure of the tests.

6.8 Performance

Performance should not be an issue for this project. However, if this were production ready, then performance would be a huge issue. Making sure the build server is not overwhelmed with too many tests or having too many people trying to contact the repository at one time could break the federation. This might happen around major deadlines, once a day when every test is run, or right before the code is put out into production.

7. Prototyping

The build prototype runs using MSBuild. Getting it to work on Visual Studio 2017 was quite difficult because the environment had to be set up correctly. The way it works is by creating a ConsoleLogger to display the logs on the console and creating a Dictionary<string, string> to put into a BuildRequestData that is later used to build the project. BuildParameters are used to add the GlobalProperties and loggers to give to the BuildManager, which builds the solution.

This only works for a .sln file. In the final build server, this will not suffice, as it works in its own way behind the scenes. I will implement a build server that can do more as stated above.

The screenshots show that solution built successfully in .95 seconds. There is normal build log output shown on the screen because I used the ConsoleLogger. I could have had it send the logs to a file using FileLogger. I needed to get the logs to output to the console, so reading from the file would add an unnecessary additional step at this point.

```
C:\WINDOWS\system32\cmd.exe
Build started 9/13/2017 5:49:09 PM.

Project "C:\Users\cmtob\OneDrive\Documents\CSE681\BuildPrototype\BuildPrototype.sln" (Build target(s)):

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

    Project "C:\Users\cmtob\OneDrive\Documents\CSE681\BuildPrototype\BuildPrototype.sln" is building "C:\Users\cmtob\OneDrive\Documents\CSE681\B
uildPrototype\Build_Prototype\Build_Prototype.csproj" (default targets):

    Project file contains ToolsVersion="15.0". This toolset may be unknown or missing, in which case you may be able to resolve this by installi
ng the appropriate version of MSBuild, or the build may have been forced to a particular ToolsVersion for policy reasons. Treating the project a
s if it had ToolsVersion="4.0". For more information, please see http://go.microsoft.com/fwlink/?LinkId=291333.
    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:
        Build_Prototype -> C:\Users\cmtob\OneDrive\Documents\CSE681\BuildPrototype\Build_Prototype\bin\Debug\Build_Prototype.exe

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

Time Elapsed 00:00:00.95
Build Process Completed
Press any key to continue . . .
```

Screenshot of Output

This prototype taught me about MSBuild and how finicky C# can be. I now know that I do not want to implement the build server using MSBuild. It does not allow for additional languages to be compiled and requires an .sln file.