|  |
| --- |
| **CSE 681- SOFTWARE MODELLING AND ANALYSIS** |
| **Remote Build Server** |
| **PROJECT-4 OPERATIONAL CONCEPT DOCUMENT** |

|  |
| --- |
| RISHIT REDDY MUTHYALA    SUID: 431658501 |

Contents

1. [EXECUTIVE SUMMARY: 3](#_Toc500364182)

[INTRODUCTION: 5](#_Toc500364183)

1. [USERS and USES: 8](#_Toc500364184)
2. [STRUCTURE AND PARTITIONING: 9](#_Toc500364185)
3. [CRITICAL ISSUES: 17](#_Toc500364186)
4. [Deficiencies I feel like in project 4: 19](#_Toc500364187)
5. [CONCLUSION: 19](#_Toc500364188)
6. [REFERENCES : 20](#_Toc500364189)
7. [APPENDIX 20](#_Toc500364190)

# EXECUTIVE SUMMARY:

In software companies, code written for any software development has millions of lines of code and it is integration of code written by several developers. So, each developer writes certain modules of code which will be tested by them and then integrated. But, when the code is integrated the build might not be successful and derive to some build errors like forgot to add 3rd party library and so on which causes the company lot of time and money. These kinds of issues lead to “CONTINOUS INTEGRATION”. Continuous integration (CI) is the practice of merging all developer working copies to a shared mainline several times a day. When new code is created for a system/software we build and test the code in context to another code and as soon as it passes the tests it is checked into the system as a part of current baseline.

The purpose of the document is to provide implementation details of build server application.

In this process of continuous integration or Build automation there are several services which are necessary to efficiently support Continuous integration. Of these services, we are now focusing on “BUILD SERVER”. Build server based on test request it downloads all the necessary source code files, meta-data information from the repository servers and then start building executable images or libraries. Build server also maintains baseline code executable images or libraries. This helps managed service team in deploying project to development server, stage server and production servers.

The Build Server will function as one of the principle components of a Software Development Environment Federation, the others being Repository, TestHarness, and Federation Client. Building these other Federation parts is beyond the scope of this development.

The development of REMOTE BUILD SERVER is capable of building multiple C# files using process pooling and on remote servers locations. This implementation is achieved in three stages.

The first, Project #2, implements a local Build Server that communicates with a mock Repository, mock Client, and mock TestHarness, all residing in the same process. This gives the developer basic design idea like how he needs to implement.

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

Finally, the third stage, Project #4, completes the whole build server process where mock repository, mother builder, different child builders , Client Gui, test harness run on different ports communicate through message passing communcation provided by COMM service.

The final product consists of a relatively small number of packages. For most packages there already exists prototype code that show how the parts can be built. For this reason, there is very little risk associated with the Build Server development.

The key features of this system can be illustrated as follows:

1. **Test Executive**: This module will demonstrate the user the working of the entire system and ensure that all the requirements are met.
2. **Client GUI**: This package will be build on wpf where there will be client gui for interaction which communicates with other servers on message passintg comunication.
3. **Repository**: Repository is responsible for creating build requests and storing them in Repository storage. It is responisble for sneding required files for child builders.
4. **XML Parser:** The parser is used to parse the test request sent by mock repository crated by client. It parses the text request and loads required source files from the repository server.
5. **Mother Builder:** Mother builder is responsibe for creating child builders on command and it is responsible for sending build requests whenever child builder is ready.
6. **Child Builder-Build Server:** Solution builder is responsible for building the code which is required for testing where the source code is fetched from the Repository server using XML parser.
7. **Test Harness:** Test harness is responsible for testing the dll files sent from the child builder. It also sends the result back to Client GUI.

The primary users of BUILD SERVER are Developers, Quality Assurance members, Managers, Instructors, Teaching Assistants etc. A few critical issues shall be considered while developing the system and their solutions would be provided. Some of these are:

* Ease of Use
* Performance
* Handling multiple test requests
* Accuracy
* Security
* Logging

Few other issues which are to be considered during development are File access issues, Maintainance issues, Data integrity issues.

This document shall explain all the functions of the system, users at various levels, application activities, interactions between various packages, modules and the structure of the build server in detail. It will also provide discussion on the various critical issues and the solutions recommended for these issues.

**Advantages:**

The advantages of build automation to software development projects include

1. A necessary pre-condition for continuous integration and continuous testing
2. Improve product quality
3. Accelerate the compile and link processing
4. Eliminate redundant tasks
5. Minimize "bad builds"
6. Eliminate dependencies on key personnel
7. Have history of builds and releases to investigate issues
8. Save time and money - because of the reasons listed above.

## INTRODUCTION:

**Remote Build Server:** A build server, also called a continuous integration server (CI server), is a centralized, stable and reliable environment for building distributed development projects. Build servers primary purpose is to generate build artifacts through activities like compiling and linking source code. Build automation is considered the first step in moving toward implementing a culture of Continuous Delivery and DevOps. Build automation combined with Continuous Integration, deployment, application release automation, and many other processes help move an organization forward in establishing software delivery best practices.

Few tasks performed by REMOTE BUILD SERVER are:

1. Parsing the xml test request sent by Client.
2. Creates number of child builders on command
3. Creates temporary document folder and builds the required source code.
4. If the build is successful then sends it to Test Harness.
5. Generates a logger file.
6. Maintains the versions of code.
7. Maintain log for all the errors and warning.
8. On kill command asks the child builders to exit.
   1. **APPLICATION OBLIGATIONS:**

The key feature of remote build server is to support Continuous Integration such that the procedure becomes less complicated, easy to use, robust, fast. The primary obligations are:

1. It should be able to parse the xml test request files.
2. It should be able to generate log file which contains time stamp, version of code, author of module, errors, warnings.
3. It should create a temporary directory where it can store required source code.
4. Sending a command to test harness to test the given code based on text request.
5. It should be able to select correct tool chain (e.g. C++, C#) on parsing the test request.
6. It should create dynamic link libraries (.dll files) through C# code for all the test request.
7. It should be able to handle multiple test request.
8. It should properly handle ready messages from child builders and build requests from client that each child builder will build each request without crashing.
9. Software collaboration federation should support continuous test and integration using Test Harness server and repository server.
10. Application should support automatic testing for running many tests efficiently.
11. All servers should be able to expose their services using WCF, so that client can able to communicate with servers.
12. All servers should be able to accept multiple requests, and run concurrently.
13. Client user interface should be designed and implemented using Windows presentation Foundation (WPF)
14. SCF should support collaboration, scheduling meetings, storing project management information
    1. **ORGANIZING PRINCIPLES:**
15. The organizing principles of the system are to perform the primary functions of Build Server and provide solutions to all the critical issues.
16. The system will be using .Net framework and Visual Studio 2017 for developing Build server application.
17. Entire architecture is designed using model view controller design pattern where model is separated from view and controller and similar with view and controller. Where any changes made to any module will have very minimal code changes.
18. Build server , Child Builder Test Harness, Repository, Collaboration structure is designed in a way that application can be able to handle multiple requests at the same time. In order to implement this feature, process pooling is adapted.
19. Followed peer to peer communication architecture to communicate between each system mentioned above using Windows Communication foundation.
20. Make use of LINQ XML for reading and parsing of test requests and use of .Net I/O features to search dynamic linked libraries and copy them to temporary directory.
21. Blocking Queue will be used to hold test requests from user.
22. User Interface is designed by using Windows Presentation Foundation provided by .Net Framework.
23. For developing maintainable code, complete user interface is developed by following MVVM pattern.
24. Command pattern is used for all the buttons in GUI.
    1. **KEY ARCHITECTURAL IDEAS:**
25. We will make use of WCF for message passing communication between all the packages.
26. We will make use of WPF for creating client GUI where we use buttons and list boxes.
27. We will make use of Linq xml for reading and parsing of test request.
28. Blocking queue is used to hold test request from server and will be helpful in handling multiple test requests.
29. Logging mechanism implementation is helpful in debugging application easily.
30. There should be builder controller which controls the building mechanism.
31. Packages should be structured such that there will be minimal code change due to change in other packages.
32. Build server should send error messages to repository server and test harness when one or more builds fail.
33. Process pooling is used to handle multiple test requests.
    1. **Transformation from PROJECT 2 to PROJECT 4:**

Project 4 is all the way transformed from project 2 by adding message passing communication using WCF and Client GUI for ease of use using WPF. In project 2 there is only one builder which can not accept multiple test requests. But in this we have process pooling where n number of child builders can be created and can handle multiple requests concurrently with out putting much load pressure on builder and increasing performance. GUI using wpf gives a nice look and feel with ease of use of application where client can build build requests and command to start n number of process and can send any number of build requests. As each server such as client GUI, Repository, Mother Builder, child Builders, test harness are run on different ports so they can be at remote locations and work based on http addresses. Project 2 did not have logging facility. But here we added build log and test logs.

# USERS and USES:

Some developers believe that Build server is Heart monitor of their software project. It can tell you when your project is healthy, and it can give you warning when your project is about to flatline. Users of Build server are Instructors, Teaching assistants, Developers, QA, Managers etc.

1. **Developers:**

Use of build server simplifies the developer’s workflow and reduces the chance of mistakes.  Your build server can take care of multiple steps such as checking out latest code, having required software installed, etc. There's no chance of a developer having some stray DLLs on their machine that can cause the build to pass or fail seemingly at random. The link between developers and QA is simplified. It makes sure developers aren't hamstrung by other developers breaking the build, and encourages developers to write better code.

1. **Quality Assurance:**

QA team is responsible for testing and deploying the source code. So build server is helpful in building the .exe and .dll files for testing of software.

Centralized builds provide easy access to code metrics -- which tests passed, which failed, how often, how well is your code covered by your tests, etc. Having a solid understanding of the quality state of the codebase reduces maintenance and testing costs by providing timely feedback that allows errors to be fixed quickly and easily. Product deployment is simplified -- the developer or QA doesn't have to remember multiple manual steps. It can be easily automated. QA personnel can go to a known location to grab latest, properly versioned builds.

1. **Managers:**

Managers make use of build results and log files generated by build server to know more about the stability of application. They can query the build results and can get idea about the position in development process. Managers use log files to estimate the release dates and budget and check whether they are in control or not.

1. **Instructors / Teaching Assistants:**

Instructors or teaching assistants can use the tool to examine whether the student’s assignments or projects are building correctly without any errors or warnings. This helps them in evaluating and grading students very easily. The can also verify how much feasible and scalable is the application.

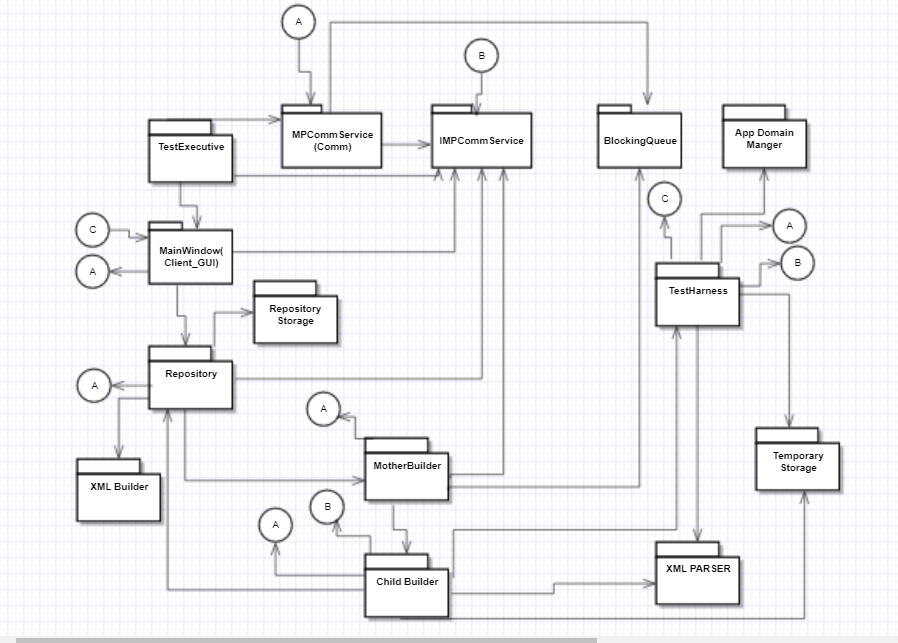
Some of other uses of build server are:

1. Scripted builds
2. Scripted tests
3. Machine independence
4. Continuous integration
5. Daily and weekly builds
6. Automated releases
7. Building in multiple environments
8. Static and dynamic analysis

# STRUCTURE AND PARTITIONING**:**

Most of the software projects are very large and contains several lines of code and several modules. So, when there are changes in requirements which might affect every module if they are not handled properly. We follow MVC architecture where we divide the source code into distinct set of packages where changes made in one package may affect other packages only a bit. Instead, the application is developed as a collection of packages. The high-level functionalities of the application are broken down into smaller tasks that must be carried out, to implement those functionalities. Logically similar sets of tasks are grouped together, into what we call packages. The various packages interact with each other, to provide the fully functional software. The chief advantages of such an approach are easier development, testing and debugging.

The Package structure of the BUILD SERVER application is shown in Figure 1. The Package Diagram is followed by a brief description of each individual package, and its interactions with other packages.



1. **Test Executive:** This is used to demonstrate Requirements and automate the procedure for demonstrating requirements. This is run on port number 8090. It starts the requirement demonstrating procedure by showing that we can build requests, send build requests to mother builder, ask mother builder to start number of child builders, and ask that to kill the child process. It creates a new comm object and receiver thread.
2. **Client GUI:** This package uses WPF for building the GUI and communicates using comm object which is run on port number 8080. It gives more interactive design the project which makes use of application easy. On build request it shows the path of xml file where it is built and user can start number of process, can send number of build requests, can also kill the child builder process. It creates a new comm object and receiver thread.
3. **Repository:** Repository is used in handling the storage of files. It is run on the port number 8081. Following are the tasks performed by repository:
4. Repository stores all the build request.
5. It sends the files to client GUI when requested.
6. On command it sends the build requests to mother builder.
7. On command it builds a request xml file with the selected files.
8. On command sends files to child builder for build process.
9. It stores the build logs and test logs.

Repository shall accept the parsed test request file. It should collect all the required source code and dependent libraries from file manager and send them to the build controller. It should also accept build logs from build controller and test logs from test harness. It creates a new comm object and receiver thread.

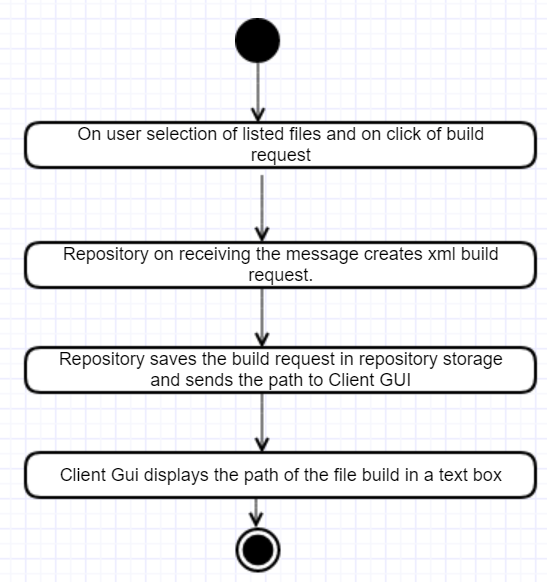
1. **XML Parser**: This xml parser parses the test requests which are in the form of xml files. This package uses the LINQ XML for parsing the xml file and sends that to repository. The repository so gets the parsed xml files which has the list source codes and libraries which are to be build and tested.
2. **Blocking Queue**: Blocking Queue package implements a generic thread safe queue. Used for communication between peers. The Blocking Queue package provides a mechanism for the XML files to be tested one at a time. It consists of two operations:

Enqueue: The enqueue operation adds all the XML files to the queue when the Build server calls the Blocking Queue package.

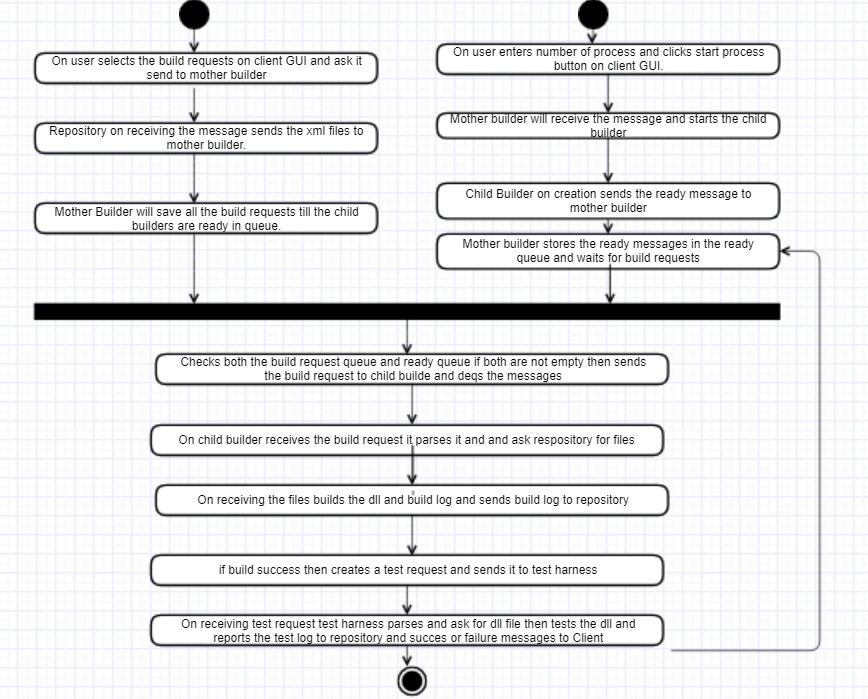
Dequeue: Once all the XML files are enqueued in the blocking queue, one XML file is dequeued at a time. If the queue is empty when the reader attempts to dequeue an item then the reader will block until the writing thread enqueues an item. The blocking queue is implemented using a monitor and lock to make the waiting efficient. The dequeue operation calls the XML parser package for decoding the XML files

1. **Mother Builder:** Mother Builder is the one core part of build server. Mother builder is run on port number 8083 and is responsible for starting child builders on different ports handling multiple request and multiple ready messages from child builders. Mother builder sends the build request whenever a child builder is free. It process class to create new process for each child builder assigning port numbers from 8084 and adding +1 for every child builder.
2. **Child Builder:** Child builder is responsible for actual build process. It uses csc command to build the files and builds log files. Then creates a test request xml file and sends it to test harness. It gets killed when receive a kill command from the client.
3. **TestHarness:** Test Harness is responsible for the actual testing of dll files. It uses reflection to load the compiled code and test them.It is responsible for creating test log and sending it to the repository. It is run on 8082 port.
4. **MPCommService:** This package has comm class which in turn creates a sender and receiver for communication between the client and server. It is dependent on IMPCommService and blocking queue for checking the messages.
5. **IMPCommService:** This package has data contract for wcf communication.

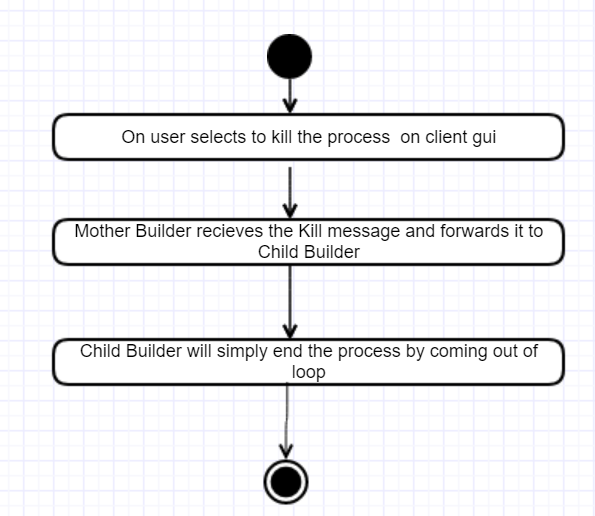
**Activity Diagrams:**

1. **Activity Diagram for Build Request:**

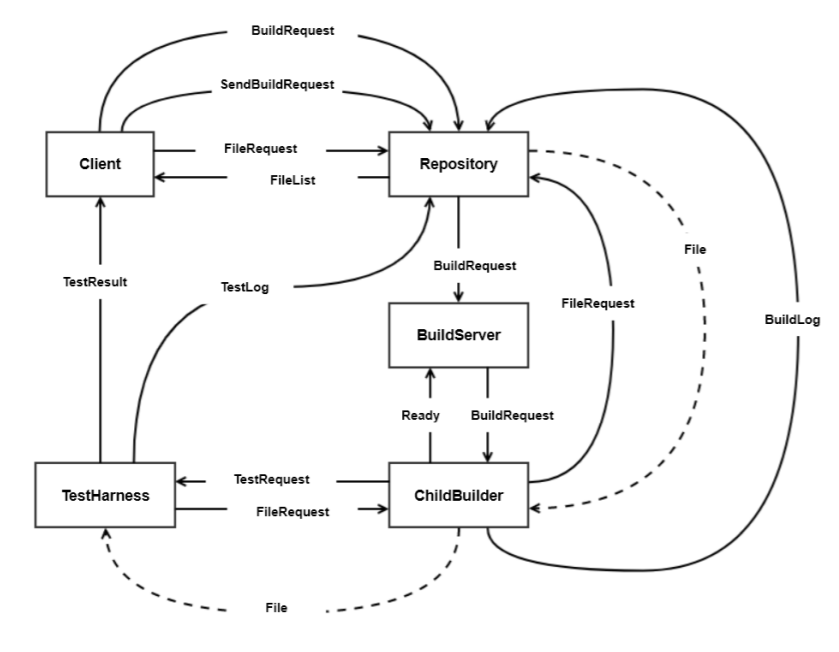
* On user selection of listed files and clicks build request client passes message to repository with selected files as arguments.
* Repository makes xml file and stores it in repo storage
* Repository replies back to client with path of build request made.
* Client gui on receiving reply message populates the build request path text box

1. **Activity diagram for starting child builders and sending build request:**

* On selecting xml files and sending them to build server.
* Build server will store them in build request queue
* It waits till there are ready child builders
* On start process click it will start the child process
* Child process on creation will send ready messages
* Mother builder will save ready messages in ready queue
* When both ready queue and build request queue have messages then sends each build request to child builder
* Child builder will parse the build request and request the files from repository
* On receiving files it should build on success should create test request and send it to test harness.
* Test harness parses the test request and asks for dll and it runs the test;
* It sends the result to client

1. **Activity diagram for killing child Process:**

* On kill button is pressed client sends message to mother builder.
* Mother builder will forward to child builder
* Child builder will stop receiving messages and ends the process by coming out.

**MESSAGE FLOW DIAGRAM: **

* Message flow diagram contains all the flow of messages in the communication process
* Filelist=message with list of strings;
* File request=commnd
* SendBuildRequest= command with list of files
* Build request=command with list of files
* Ready=command
* buildLog= post file
* test request=command with filename
* testlog=postfile

# CRITICAL ISSUES:

Critical issues with the Build server are:

1. Ease of Use:

Build server is used to make developers workflow easy and make the process of automation simpler so that the project is more efficient and accurate. In this aspect Ease of use is very important concern for the developers. We should take care that we don’t build a rigid application such that it only works in few cases. Build server can build lot of test libraries without the intervention of human. So, to achieve an effective Build server it should be simple and easy to use.

Solution: The code structure should be well defined various use cases and their impact should be considered beforehand. There should be very user-friendly interface for build server application. We are not building GUI for our 2nd project where most of the commands are send through the test executive. In 4th project we will be creating a GUI and client can send any test request that are already on the memory.

1. Performance and Productivity:

Build server is going to be very busy in the final stages of deployment. Performance and productivity of build server are two highly concerned issues for build server. We can measure the performance by time taken for each build.

Solution: To decrease the complexity our packages should be in such a way that are interacting with each other effectively without any dead lock situation. Sharing of resources is also a good concept. While handling multiple test request multi-threading is a good concept to be followed.

1. Demonstration:

Demonstration of requirements by showing the graders and instructors that we met all the requirements is an issue. To project the proper working of the system, the complete set of requirements need to be showcased to the user so that the user understands the working of the system in an adequate manner. Also, all the functionalities should be tested properly.

Solution:

The showcase of requirements should be done by the test executive on the console output. Every requirement should be tested and demonstrated perfectly with the success or failure of the system.

1. Accuracy:

Build server should be accurate in its results. If the build is successful it should return success or it should notify with errors. If the build server is not accurate the process of continuous integrations fails leading to whole project failure.

Solution:

The xml test request should be specific to our input style and parsing of xml of file must be accurate so that it loads the source file correctly and the build process is accurate without failure.

1. Security:

Security is the very important factor which is to be considered while developing build server. Breaches of security could include damages to the system, loss or theft of data, and compromise of data integrity.

Solution:

Users should be given specific permissions such as developers should only have scope to specific parts of build server where managers can be given scope to other specific parts of the servers by which we can maintain the system stable and secure.

1. Logging:

The logger of the system should be updated every time build happens. It should be effective enough to be accessed to client using queries. It should not lose any data. It should also maintain the version of code so that we can know how many times did we update the code with respect to change in requirements.

Solution:

An interesting way of logging messages that has been adopted in this project is by recording the results of the builds in a dictionary and print the values to the user after every test request is executed After all the test requests are executed we can retrieve a string by the getLog() method and create a log file by adding the details. We can then insert the log file in the database.

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

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

1. Building and testing both C# and C++ code.

Solution: For building, set environment variables as demonstrated in Help Session demo, and use tool chain commands for each type of source. For testing, trap exceptions on loading native code libraries in the C# TestHarness and direct to C++ TestHarness, as demonstrated in class.

1. Managing EndPoint information for Repository, BuildServer, and TestHarness.

Solution: Store Enpoint information in XML file resident with all clients and servers and load at startup.

# Deficiencies I feel like in project 4:

1. As build process now has process pooling build server performance increases, but there will be too much load built on test harness as it has only one server. So there is need for increasing performance of test harness.
2. There is no proper file accessing method. As post file is passing file in form of bytes there may be locking situations while accessing the file.
3. Need for more stable message passing communication system.

# CONCLUSION:

we know that build server is an effective way for faster releases of the project and also for the better project quality. It also reduces time losses with budget control of any software product. So, we need to design it in an effective way covering all the critical issues.

# REFERENCES :

1. https://en.wikipedia.org/wiki/Continuous\_integration
2. CSE-681 website
3. <https://stackoverflow.com/questions/1099133/what-is-the-point-of-a-build-server>

# APPENDIX

