# CSE 681: SOFTWARE MODELLING ANALYSIS

## Project # 1- Operational Concept Document

## Build Server

Instructor- Dr. Jim Fawcett

Sarath Patlolla

(SUID -764816172)

spatloll@syr.edu

09/10/2017

# Table Of Contents

# 1. Executive Summary

Build Server is a software development approach in which developers merge their code into common repositories several times. Every commit made in the code is then built. This allows the team to detect a problem early. To merge the code, every check in should build and run a series of tests that ensure common repository is always ready for deployment. It provides the early feedback for any code that is developed and merged into common repository.

Build Server means isolated changes are tested and reported when they are added to larger code base. The goal of build server is to give rapid feedback so that any defect can be identified and corrected easily.

The users of the Build servers are the,

- Software developers since they incorporate the server build process in their builds and automate the building process
- QA managers testing is one of the primary tasks which needs to be addressed in a software development after the successful build of the process automated testing is done to fix bugs.

Nowadays software are built by integrating many modules in a package. Here in the build server client specifies his requirements and submits them, the developer in accordance with the requirements develop various libraries and store them in the repository now the libraries are parsed and sent to the build server where an automated build takes place after the successful built of the libraries they are sent to the test harness where the precompiled libraries are tested for the specified test conditions by accepting the test conditions in the form of an XML file.

In the build server a temporary directory is created and all the test cases are stored in there and after the build and the testing is done the directory is deleted.

# 2. Introduction

The following is an Operational Concept Document describing the basis for development on a comprehensive Build Server. The executive summary will provide the summary with the important packages that are needed for making a build server and the flow of the libraries from one package to the other. Build server is continuously from what is committed from the repository

Build server is a tool that enables continuous integration. They are referred to as Continuous Integration(CI) servers. Even if the project has only one developer working on it, the build server adds value to in many ways such as it eliminates the effect of “it works on my machine” syndrome, in which the software only works in one environment only.

The build server starts with a clean state, so no approved configurations or artifacts are present. Since the code is pulled from the repository, only the working code would be generated in the end. This has an additional advantage of able to predict, if there are any conflicts in the code or any missing dependencies between the files

CI is a development technique that enables the developers to integrate code into a shared repository multiple times. It helps to reduce errors and bugs while implementing new features in the project. Whenever we are implementing a new feature in the code or a adding a new package to the software there is always a possibility of breaking the code. So, integrating a new feature and trying to build automatically removes any bugs in its inception stage only. And moreover, a clean repository is maintained and all the members of the team are aware of the changes that have been made in the code and that they can accommodate those features in the code which they are coding.

The idea of a build server is that it sits there and builds after every checkin, to make sure nothing broke. We can also include tests that run automatically after building, including unit tests and performance tests. The system can be configured to export a fully functional installer that can be made available on the intranet for other departments to download and use/test. This saves a lot of time and money.

The fundamental principles of a build server are

- Integrate the code multiple times: This really depends on the branch and merge policy. If everyone commits to a single branch then setup builds for that branch. If there are different developer for different branches, then create an integration branch that builds frequently.
- Provide instant feedback wherever possible: Irrespective of where the developers are committing to the code (individual or shared branches), we need to find a way of providing instant feedback, after each commit and also to ensure that blame for broken build is instantly escalated and acted upon.

# 3. Users and Uses

## 3.1) Software Developers:

The developers can run the build server for building of the packages or libraries which they have developed. With the concept of continuous integration in the build server while pulling the code from the repository the newly inducted code would be built separately without creating any conflicts with the pre-existing codes in the repository. In general when a developer develops a code there is always a high possibility that the code which he has generated would be working only on his computer ,when tried to run the same code on other systems the code might fail. The following are benefits developers enjoyed by using the build server.

- Early detection of bugs.
- Early detection of integration issues.
- Early detection of failed unit test / self-testing code.
- Automate deployment.
- Build stuff faster.

## 3.2) Software Architects:

As the name suggests the software architects manages the system architecture and its attributes. The software architects are one of the primary beneficiaries from the Build server where a continuous and automated build of the code is done by accepting the code from the repository. The continuous integration (build server) is a practice. The following are the uses of the build server for the software architects:

- Engaging operations as a customer and partner, “a first-class stakeholder”, in development.
- Engaging developers in incident handling. Developers taking responsibility for their code, making sure that it is working correctly, helping (often taking the role of first responders) to investigate and resolve production problems.

- Ensuring that all changes to code and configuration are done using automated, traceable and repeatable mechanisms – a deployment pipeline.

## 3.3) Quality Assurance Manager:

The Quality Assurance Managers are also one of the users of the automated build server. The QA managers have the responsibility of determining, negotiating and agreeing on in house quality procedures, standards and specifications, assessing customer requirements and ensuring that are met. To the QA team, the benefits of having a build server are immediate and abundant , ameliorating challenges typically associated with the code.

- Test packages, hosted in the QA repository, can be maintained by multiple QA personal and automatically executed when new code is deployed .With new found software defect becoming an automated test case.
- Cut down on overhead and testing time while improving effectiveness and reliability of test cases.

## 3.4) Teacher Assistants(TA's)

The TA's works closely with the developers to check whether the system works in an expected fashion without any issue and also make sure the code is clean and simple.

## Uses:

The Build Server is deployed in many organisations because it has the following advantages over the conventional process

a) Run the tests in multiple platforms: By building the code on a build server the code can run on any computer. It eliminates the "Run only on my computer" syndrome.
b) Increase the code coverage: A build server can check the code for code coverage, every time we build a new code or add additional functionalities

to the code, the code would be sent to the test harness where with the given test cases testing would be done and verified whether it meets the client's requirements or not.

c) Build Code Faster: With parallel build support, the tests can be split and build processes into different machines, so the building and the testing would be done faster than when done at the local machines. It will also consume less local power and resources so we can continue other things when the building would be going on.

d) No conflicts with other packages: By making the code isolated from rest of the packages and building will reduce any conflicts that might arise.

e) Decrease Code review time: By having a Build server and Version Control Server communicate with each other and to give a feedback as to when a merge request is good to merge. It can also show how the code coverage would be affected by it. This can dramatically reduce the time it takes to review a merge request.

# 4).Structure and Partitioning

The modern software's are very large and it is impractical to develop them in a lone source file. 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. These packages generally act as libraries and provide the required functionalities.

The package diagram of the Build server is shown in the figure and is followed by a brief description of each individual packages, and their interactions with other packages.

A package diagram is a Unified Modelling Language that depicts the relations between the different packages that make up the model.

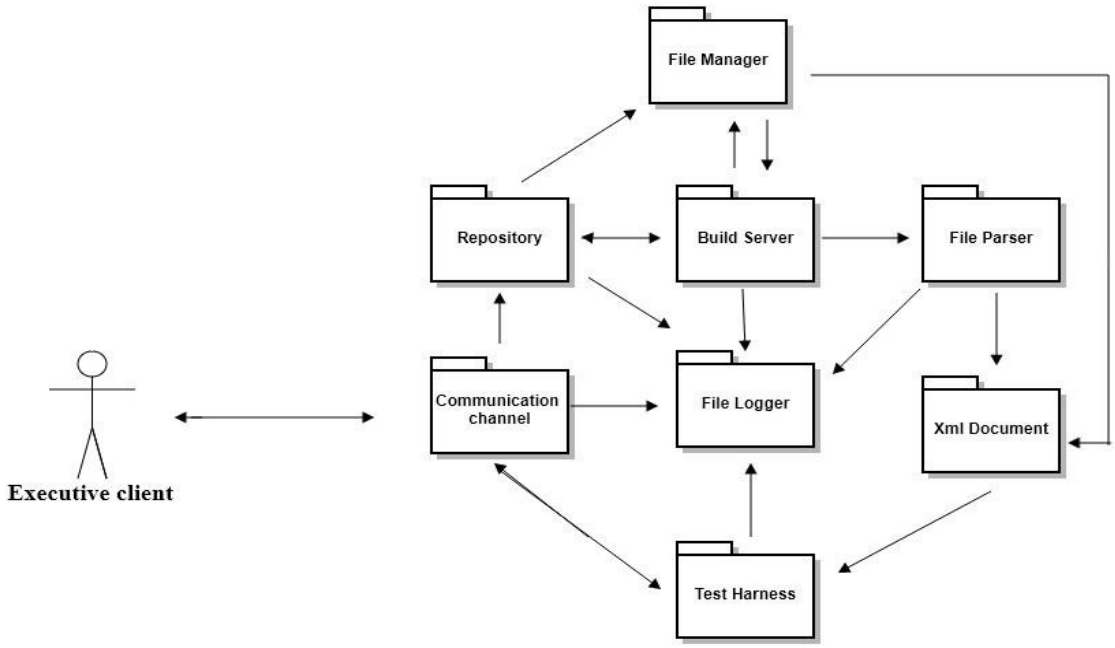



Figure: Package diagram of the Build Server

The working and the interactions of the various packages is given below:

4.1) Executive Client: The client is the end user who specifies his requirements, about the software which he needs. The client is the end user in the software development process. The client interacts with the development, testing and the QA team through a communication channel where he specifies his requirements.

4.2) Communication Channel: A formal communication channel is an 'official' means of disseminating information in the organization hierarchy. In other words, it's any system by which you can communicate with the various departments in the organisation.

4.3) Repository: Repository is a place where the data is stored and organized in an efficient way. Here it is the baseline which holds all code and document for the current baseline, along with their dependency relationships. It also holds test results and may cache in test images. The repository is in direct access to the client via communication channel.

4.4) File Logger: File logger is to keep the logs of all the activities whether the build was done correctly or, interactions with the client or any other package interactions, test results of whether the test was executed perfectly as expected or not. It can help in communicating with the system, and thus reduce the complexity of the system. Thus, logger is seen at every interaction in the system.

4.5) Build server: A build server is a tool that enables Continuous Integration, an excellent practice to follow when developing software. A build server, also called a continuous integration server (CI server), is a centralized, stable and reliable environment for building distributed development projects. Based on build requests and code sent from the Repository, the Build Server builds test libraries for submission to the Test Harness. On completion, if successful, the build server submits test libraries and test requests to the Test Harness, and sends build logs

to the Repository. The Build server is in direct interaction with the File logger and submits its log details to the file logger about the successful building or not of the test libraries.

4.6) File Manager: The File manager is used to move files from one package to the other. The file manager itself doesn't have any data it uses the help of repository to move the data to the build server and after the successful build the libraries are moved to test harness via file manager.

4.7) File Parser: Parsing is the process of discovering and classifying the parts of some complex thing. In this context parsing is the process of some form of syntactic analysis. There are a lot of reasons you may wish to parse source code beyond compiling its text. For example:

- Building code analysis tools
- Searching for content in or ownership of code files
- Evaluating code metrics
- Compiling "little embedded languages"

4.8) XML Document: Repository in our application sends test requests, in the form of an XML file that specifies the test developer's identity and the names of a set of one or more test drivers with the code to be tested. The .xml file path is passed to the test harness as a command line argument.

Following is an example of the test request sent by the XML reader to the test harness:

<? Xml version="1.0" encoding ="utf-8"?>

<testRequest>

<author>Sarath Patlolla</author>

<Test name="First Test">

<test>test1.dll</test>

<test>test2.dll</test>

<test>test3.dll</test>

</Test>

</testRequest>

4.9) Test Harness: Runs tests, concurrently for multiple users, based on test requests and libraries sent from the Build Server. Clients will checkin, to the Repository, code for testing, along with one or more test requests. The repository sends code and requests to the Build Server, where the code is built into libraries and the test requests and libraries are then sent to the Test Harness. The Test Harness executes tests, logs results, and submits results to the Repository. It also notifies the author of the tests of the results.

# 5).Application Activities

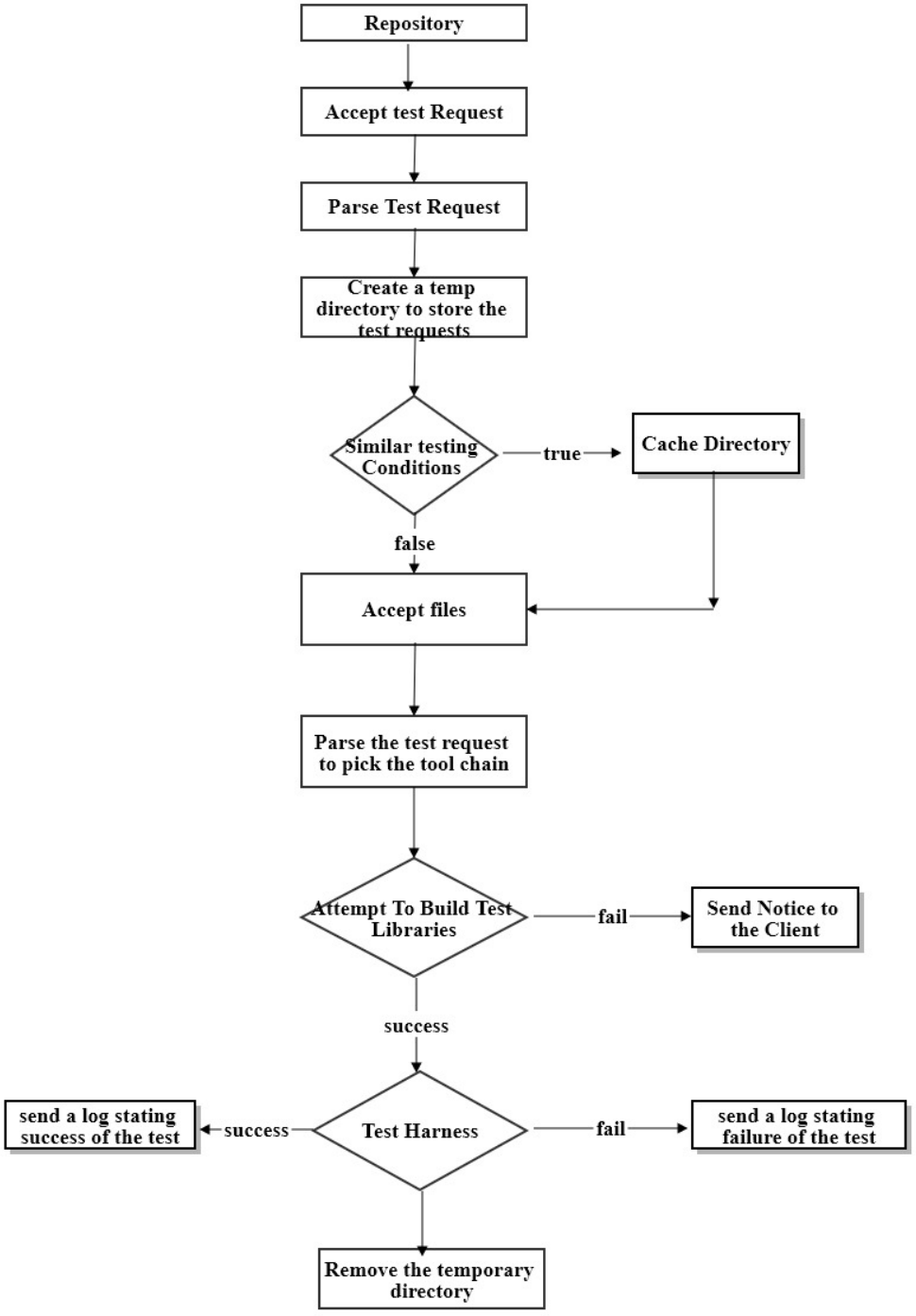



Figure : Activity diagram in a build Server

The Activity diagram above depicts all the activities that are taking place in a build server. Here once the source code has been written by the developers and stored in the repository along with the test conditions, the following are the activities that are taking place in a repository,

- The repository sends a test request to the build server regarding the test conditions.
- The build server accepts the test request and parses the test request. Parsing of test request means dividing the test request into logical and semantic blocks of code and deciding what files to be sent to the build server.
- The build server accepts the build log and the test log both at the separate time because first the build process needs to be done and then only can the test be done.
- On command, the build server sends the build log to the client and the it sends the test log to the logger.
- We have a dedicated file logger which is to send files from one package to the other.
- After parsing the test request to the build server, a temporary directory is created in the build server, the temp directory could be given a unique name such as author name followed by time and date to each directory to maintain them unique.
- The temporary directory is to store the test request of the file.
- Build server can also create a cache directory so that many files which are tested under similar conditions would be grouped together.
- The build server then accepts the request which was sent from the repository.
- Parse the test request to pick the tool chain (if the extension is .cs then use c# compiler or if the extension is .cpp then use c++ compiler.
- Attempt to build the test libraries. The different libraries are build separately because the build might cache some info.
- If the build fails then a notice is sent to the client stating the failure of the build and specifying all the errors and warning messages that have been generated.
- If the build is succeed then it is sent to the test harness.
- The test conditions are sent to the test harness in the form of an XML file.
- The temporary file is then removed.

# 6).Critical Issues and Suggestions

Achieving widespread use of a tool and reusability of complex software components requires a concerted focus on the critical factors that might be potential blocks during the analysis of the architecture. Identifying the critical issues allows a good understanding of any major pitfalls that might occur during the design, development and maintenance of the system. The following are some of the critical issues and some suggestions to overcome them:

## 6.1) Long Builds:

As more and more packages are added to the Software, more lines of code are added to the package and testing to be done on those packages increases. This results in:

- Longer wait times for the developer for the builds to complete, which is somehow time lost.
- Context switching: The developers lose the context they are working on and they lose interest.
- Product quality is lowered: When a developer commits changes and breaks something, it would be discovered only after the build runs. The longer the build, the lesser bugs that could be fixed.

### Suggestions

Since the build time for the long build is high, to increase up the build time we

- Run builds in a machine
- Running only those parts of the build, which have been changed
- Moving the entire disk operation to the RAM and eliminating the I/O
- Precompiling the shared dependencies that change less frequently

## 6.2) Large volume of Builds:

When there is a single package to be build, it would run quickly, but as the size of the organization increases more and more developer teams would be working on the development, and they would be all submitting their packages to the build server and this would sometimes lead to the jamming of the packages at the server. This has an influence on:

- Developer who needs to multiple packages as more packages get jammed at the server the time taken to build all the packages would increase.
- Limited resources: the organization which is running large numbers of builds, will have limited access to build servers during specific time windows, or the servers are often overloaded and builds will take much longer.

## 6.3) Complex Builds:

Software projects are built by adding many federation packages(libraries) and are built by different teams. As the project evolves there are multiple versions of the same project which is developed. Complex build reduces the flexibility of the software and makes it difficult to build. It has the following effects:

- Complex builds are brittle i.e maintain them is very difficult since they introduce bugs due to partial or incorrect sources.
- With complex builds due to partially specified dependencies an incremental run could break the build and teams are forced to run the entire build in all scenarios.
- Complex builds are generally long.

Suggestions

These are more “heavyweight” solutions that can have a bigger impact on long and complex builds, but require more effort:

- Distributed builds – parallelizing builds across a cluster of build servers
- Manually partitioning Make files – breaking the build into smaller components

- Optimizing Make files – rewriting Make files to make them run more efficiently
- Unity builds in C/C++ – combining source files into one file to reduce file access

# 7). Conclusion

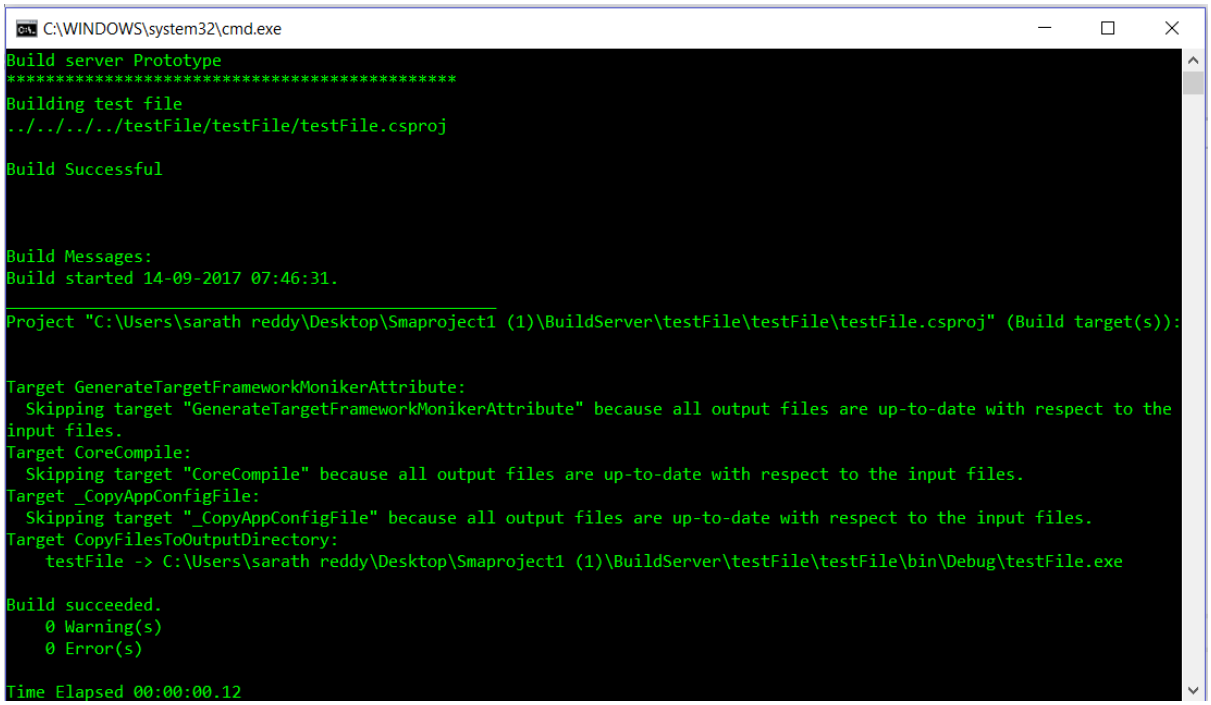
Build server can speed up the software modelling process by accommodating for multiple builds at a single time and also the errors and bugs in the code can be identified at an early stage in the development process. By using the build server for building of the code there is a dedicated server whose focus is only to build libraries or the executable files which run on any system. Thus by being careful about some critical issues which might arise in the building of the source code process, the server build can be very helpful to an organization.

# 8). References


1) https://ecs.syr.edu/faculty/fawcett/handouts/CSE681/Lectures/Project1-F2017.htm
2) https://en.wikipedia.org/wiki/Build_automation
3) https://en.wikipedia.org/wiki/Continuous_integration
4) https://stackoverflow.com/questions/6511380/how-do-i-build-a-solution-programmatically-in-c

# Appendix

Here we are building a prototype in c# using visual studio. The prototype runs on MS build. Here we are providing the test file path as the arguments to build it. On successful build of the server it accepts the test file as an arguments and then build process is initiated.




The test file was successfully built using the MS build prototype.