Project #5 – Continuous Build and Integration System (CBIS)

Preliminary Architectural Concept

Version 1.1

(new version updates concluding comments and suggested prototypes)

Jim Fawcett

11 November, 2014

Purpose of the Project:

Executing large software development projects can be challenging. Large projects may have scores or even hundreds of developers all working on a common set of goals and building hundreds of packages that have complex interdependencies. Even with diligent and effective management there will be many opportunities for miss-communication and developers may have inaccurate assumptions about the interfaces and functionality of packages that are developed by others but on which their code depends. These problems are exacerbated if there are significant delays between the development of code and its incorporation into the rest of the developing system, which we shall refer to as the current software baseline.

The primary goal of this project is to develop an architecture for a software and hardware environment in which developer's code can be continuously integrated with the rest of the developing software baseline. This architecture will consist of three servers and an arbitrary number of clients:

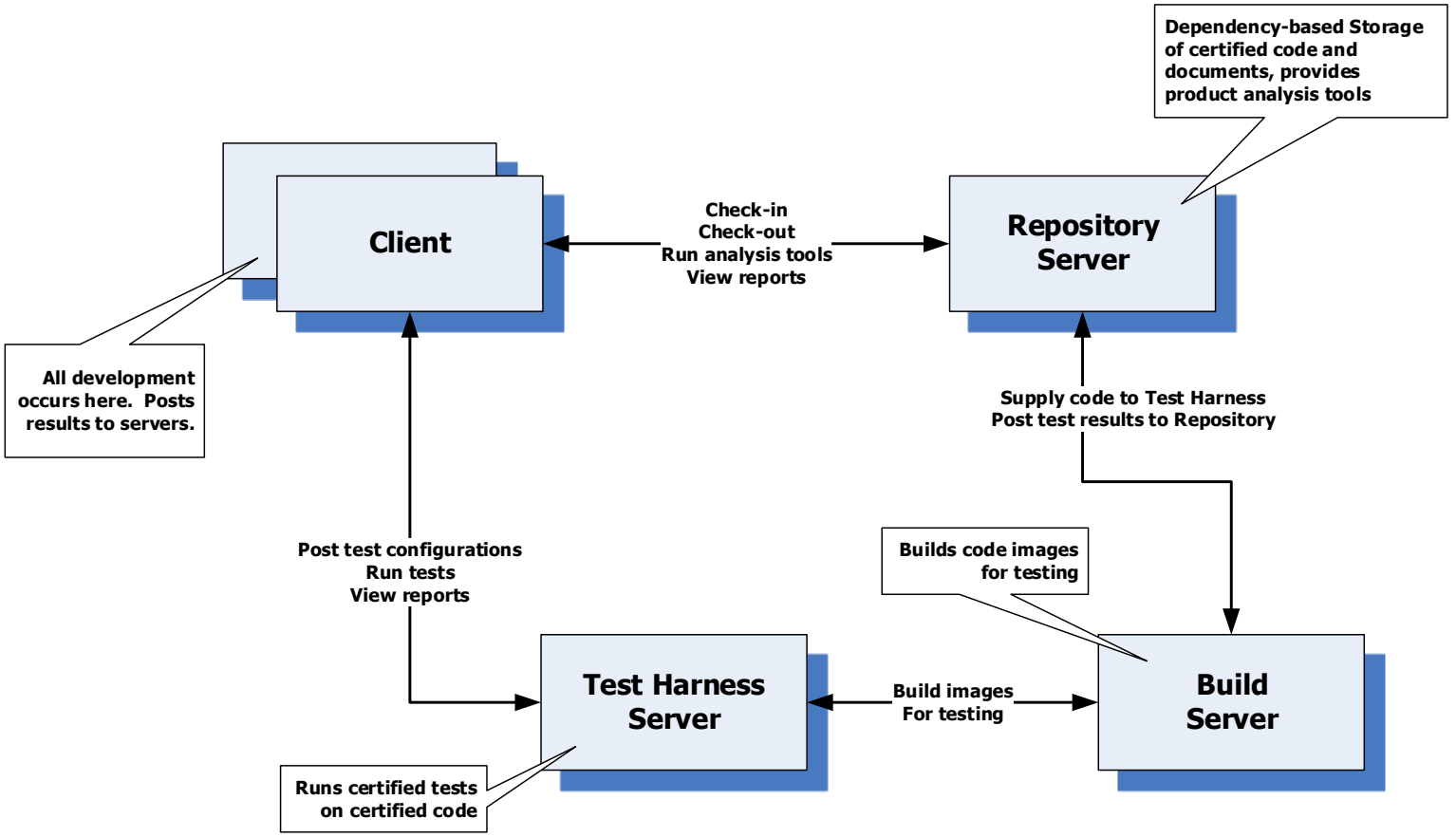
1. Repository Server:
   Stores all of the developing baseline. If a package is not stored in the Repository it is not part of the baseline. Its functions include:
   a. Defining modules as lists of packages contained in the Repository store. Also defines subsystems as collections of modules contained in the Repository store.
   b. Maintaining the integrity of modules which are collections of packages accessed through an interface and object factory. Once an interface and object factory have been defined they cannot be changed without approval of the Project Manager or Software Architect. Each module is developed by one team of developers.
   c. Storing test suites source code and associating each test suite with a module or subsystem. Subsystem test suites may contain test suites of its individual modules and may contain additional test code as well.
   d. Managing check-in and check-out of packages source code to and from modules.
   e. Analyzing the dependency of modules and subsystems and recording the dependencies with XML metadata.
   f. Executing queries about dependency relationships.
   g. Storing packages that are not currently part of a module. These orphan packages are not considered part of the current baseline until they become part of a module, either by creating a new module or by adding to an existing module.

2. Build Server:
   Compiles any code needed to execute tests, based on request messages from the Test-Harness Server. Its functions include:
   a. Cache source code modules previously used for builds.
   b. Accept build request messages from the Test-Harness Server.
   c. Request from the Repository Server module source code needed for each build that have changed or been newly added.
   d. Build images as requested and upload to the Test-Harness server if build is successful.
   e. Send build error messages to the Repository and Test-Harness Servers if one or more builds fail.

3. Test-Harness Server:
   Holds test suites for each module and subsystem and executes them on demand. Its functions include:
   a. Maintain a cache of compiled modules that represent the current baseline.
   b. Load test suites defined by XML messages.
   c. Request modules from the build server that have changed since last cached or been newly added to the baseline. These should be loaded based on dependency information maintained by the Repository.
   d. Compile the current tests and execute, returning success notification, error notifications, and results log.
   e. Send selected notifications to registered developers for all modules that depend on the tested modules.

4. Client:
   Provides a user interface into the CBIS system for developers and managers. Its functions include:
   a. Create and send messages to the Repository and Test-Harness Servers.
   b. Receive and display messages from the Repository and Test-Harness Servers.
   c. Upload source modules to the Repository Server for check-in.
   d. Download source modules from the Repository Server for check-out.
   e. Download test messages and result logs from the Test-Harness Server.

Figure 1. Continuous Build and Integration System



Requirements:

Your CBIS Architecture document:

1. **shall** be prepared as a Microsoft Office Word file, using embedded Visio diagrams.
2. **shall** partition CBIS processing into subsystems. You should partition at a more detailed level for critical subsystems.
3. **shall** explore and describe the user interface(s) you will provide and interfaces between each of the subsystems.
4. **shall** describe the uses/responsibilities, activities, events, and interactions of each of the partitions in your architecture.
5. **shall** use both text and diagrams for the descriptions in 4, above.
6. **shall** prepare a software prototype of a cache for dependency-based builds in the Build Server.

You are invited, but not required to prepare additional software prototypes to help you explore the CBIS architecture and its issues. Please document prototype code you develop in an Appendix. Note that prototype code should effectively support your conclusions and recommendations concerning the Continuous Build and Integration System. Prototypes will be judged by the effectiveness of their design and implementation, the usefulness of their output to convey information about CBIS, and the strength of the conclusions you draw, based on the prototype(s).

Additional Prototypes:

Below find some suggested prototypes. Some of these are fairly simple to implement, some are quite challenging[1].

1. Dependency graphs: Define a graph class, create an instance that represents some part of the Project code (you can use your Project #2 code to help populate the graph) and plot the resulting graph.
2. File Cache Manager that plugs into the communication system. This is an extension of the Build Server's Cache prototype that you are required to develop.
3. Project metadata generator and scanner for the Repository Server.
4. Build analyzer uses dependency analysis to define all files needed to support a given test driver.

Concluding Comments:

This project statement has provided an initial structuring of the CBIS system. You are asked to review this structure and make any revisions that seem appropriate. Then you are to explore the structure and activities of each of the services defined above, and consider any additional services that would improve the usability and effectiveness of the system.

[1] The more challenging prototypes would make excellent Master's Projects for CE students.

This initial concept is virtually silent about:

1. Use cases for baseline updates and testing.
2. Suitable user interfaces.
3. The mechanics of file transfer and caching.
4. Details of the communication system, message catalog, notifications.
5. Services provided by a Message Handler, Query Handler, and how and when notifications are required.
6. Performance issues associated with running test suites spanning the entire software baseline.
7. Process of initiating tests, building test suites, returning results, storing results.
8. Uses of type and package dependency analysis.
9. Scaling out the CBIS system through use of multiple Repository, Build, and Test-Harness Servers.

There are more questions about these services and about the other services which will be left for you to enumerate and explore.

References:

1. C# High Resolution Timer, based on performance counters, http://www.ecs.syr.edu/faculty/fawcett/handouts/CSE681/code/HiResTimerInterop/