Project #4 - Remote Build Server

Version 1.2, Revised: 11/29/2017 15:12:53
Due Date: Wednesday, December 6th

Purpose:

One focus area for this course is understanding how to structure and implement big software systems. By big we mean systems that may consist of hundreds or even thousands of packages and perhaps several million lines of code.

In order to successfully implement big systems we need to partition code into relatively small parts and thoroughly test each of the parts before inserting them into the software baseline. As new parts are added to the baseline and as we make changes to fix latent errors or performance problems we will re-run test sequences for those parts and, perhaps, for the entire baseline. Because there are so many packages the only way to make this intensive testing practical is to automate the process. How we do that is related to projects for this year.

The process, described above, supports continuous integration. That is, when new code is created for a system, we build and test it in the context of other code which it calls, and which call it. As soon as all the tests pass, we check in the code and it becomes part of the current baseline. There are several services necessary to efficiently support continuous integration, as shown in the Figure 1., below, a Federation of servers, each providing a dedicated service for continuous integration.

click to zoom

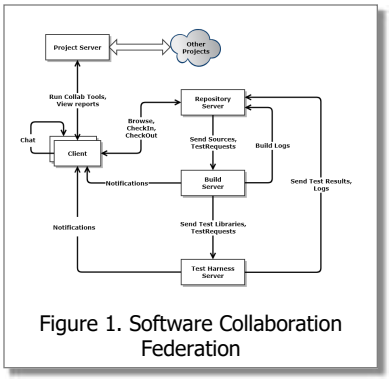


Figure 1. Software Collaboration Federation

The Federation consists of:

- **Repository:**
  Holds all code and documents for the current baseline, along with their dependency relationships. It also holds test results and may cache build images.
- **Build Server:**
  Based on build requests and code sent from the Repository, the Build Server builds test libraries for submission to the Test Harness.
- **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.
- **Client:**
  The user's primary interface into the Federation, serves to submit code and test requests to the Repository. Later, it will be used view test results, stored in the repository.
- **Collaboration Server:**
  The Collab Server provides services for: remote meetings, shared digital whiteboard, shared calendars. It also stores workplans, schedules, database of action items, etc.

Build Server:

In the four projects for this course, we will be developing the concept for, and creating, one of these federated servers, the Build Server - an automated tool that builds test libraries. Each test execution, in the Test Harness, runs a library consisting of test driver and a small set of tested packages, recording pass status, and perhaps logging execution details. Test requests and code are submitted by the Repository to the Build Server. The Build Server then builds libraries needed for each test request, and submits the request and libraries to the Test Harness, where they are executed.

The four projects each have a specific role leading to the final Remote Build Server:

- For Project #1 you will create an Operational Concept Document (OCD) for a Remote Build Server, and a small prototype demonstrating programmable builds.
- Project #2 focuses on building the core Build Server Functionality, and thoroughly testing to ensure that it functions as expected.
- In Project #3, you will build prototypes for Process Pools, Socket-based Message-passing Communication, and for a Graphical User Interface (GUI), packages all needed for the last project. These are relatively small "proof-of-concept" projects in which you experiment with design and implementation strategies.
- Finally, in Project #4 you will build a Remote Build Server, using parts you developed in earlier projects. You will also add design details to your OCD, from Project #1, to create an "as-built" design document.

So, for this project #4, we will develop a Remote Build Server and document its design with a document based on the Project #1 OCD. Your Build Server should:

T N P B

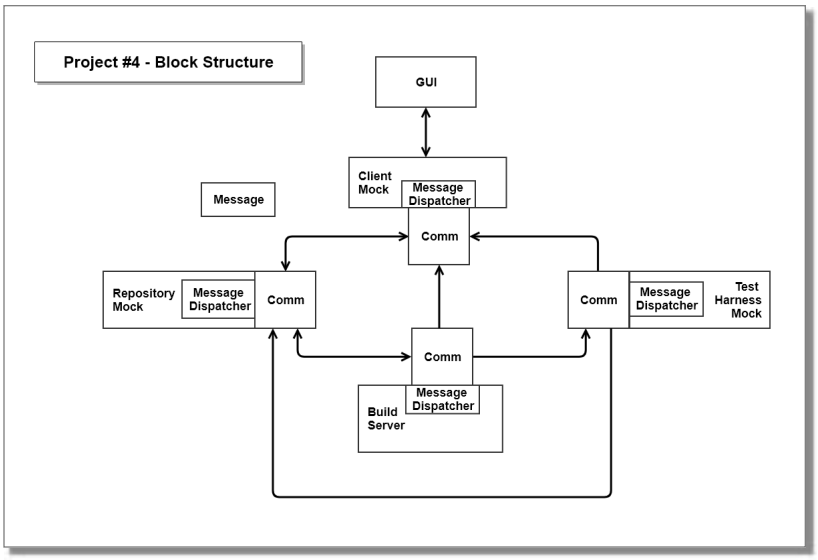
- Provide a Build Server that uses Message-Passing Communication based on your Comm prototype.
- Use mock Repository and Test Harness servers that are functioning processes, with message-passing communication. However, they provide just enough functionality to demonstrate that your Remote Build Server functions as expected.
- Provide a mock Client process, using WPF, based on your GUI prototype that has just enough functionality to demonstrate that your Build Server functions as expected.

Your Build Server Design Document should:

- Build on the Project #1 OCD, keeping its basic structure, but elaborating with design details.
- Show activity diagrams, package diagrams, and class diagrams that illustrate the way you've implemented your server and its environment.
- Comments on possible inadequacies and errors of commission.
- Draws conclusions about what you like and don't like about your final implementation.

Note that, in these projects, we will not be integrating our Build Server with a Federation's Repository and Test Harness Servers. Instead, we will build mock Repository and Test Harness servers that supply just enough functionality to demonstrate operations of the Remote Build Server. The Build Server will use a "Federation ready" communication channel to communicate with the mock servers, and we will build a mock client that has just enough functionality to demonstrate Build Server working in this environment.

So the mock Repository and mock Test Harness are simple servers, running in their own processes, using our Message-Passing Communication, to send and receive requests and replys. However, the Mock operations are simple - not nearly as complex as full up Federated servers.



We will discuss all of this at length in class and the help sessions.

Requirements:

Your Build Server

1. **Shall** be prepared using C#, the .Net Frameowrk, and Visual Studio 2017.
2. **Shall** include a Message-Passing Communication Service built with WCF. It is expected that you will build on your Project #3 Comm Prototype.
3. The Communication Service **shall** support accessing build requests by Pool Processes from the mother Builder process, sending and receiving build requests, and sending and receiving files.
4. **Shall** provide a Repository server that supports client browsing to find files to build, builds an XML build request string and sends that and the cited files to the Build Server.
5. **Shall** provide a Process Pool component that creates a specified number of processes on command.
6. Pool Processes **shall** use message-passing communication to access messages from the mother Builder process.
7. Each Pool Process **shall** attempt to build each library, cited in a retrieved build request, logging warnings and errors.
8. If the build succeeds, **shall** send a test request and libraries to the Test Harness for execution, and **shall** send the build log to the repository.
9. The Test Harness **shall** attempt to load each test library it receives and execute it. It **shall** submit the results of testing to the Repository.
10. **Shall** include a Graphical User Interface, built using WPF.
11. The GUI client **shall** be a separate process, implemented with WPF and using message-passing communication. It **shall** provide mechanisms to get file lists from the Repository, and select files for packaging into a test library[1], e.g., a test element specifying driver and tested files, added to a build request structure. It **shall** provide the capability of repeating that process to add other test libraries to the build request structure.
12. The client **shall** send build request structures to the repository for storage and transmission to the Build Server.

T N P B

13. The client **shall** be able to request the repository to send a build request in its storage to the Build Server for build processing.

Your As-Built Design Document[2]

1. **Shall** build on your OCD from Project #1.
2. **Shall** used activity diagrams, package diagrams, and class diagrams to describe the essential parts of your design and implementation.
3. **Shall** comment on changes to the core concept as your design evolved, and on deficiencies you feel your project incorporates[3].

---

1. A test library consists of a test driver package and code packages to be tested. The test driver package contains a class, derived from a base Test class that defines an override-able bool test() function. The test driver is build by the author of the tested code, and has the information necessary to make test calls into those packages.
2. You may use alternate office suites like WPS or LibreOffice, and diagrammers like gliffy, available as a chrome app. You may also find WorkFlowy useful for organizing both documents, like an OCD, an also for thinking about code structure.
3. You won't loose points for this document by citing design and implementation deficiencies. You only loose points for not citing deficiencies I think are relatively obvious when grading your document.

What you need to know:

In order to successfully meet these requirements you will need to know:

1. A lot of details about your implementations of the preceding projects.
2. Definitions for Dynamic Link Libraries - see the class text, C# 6.0 in a Nutshell.
3. How to organize and prepare a technical document. Here's some help.