The Importance of Building Earnestly

Wednesday, 16 February 2005
Last Updated Friday, 12 September 2008

Building your application is key to a successful, repeatable, development process. A reproducible build that works at all levels allows you to proceed with confidence and be more agile. Yet many organizations (agile and not) leave the build process to chance, even though all can benefit, regardless of their method.

earnest: 3. businesslike, (not distracted by anything unrelated to the goal)[wordnet]

At Right: Tennessee Valley Authority. Construction of Douglas Dam. June 1942.
Photo by Alfred T Palmer.

Bootstrapping the Development Process

bootstrap: transitive verb: to promote or
develop by initiative and effort with little or no assistance.(www.m-w.com)
A software development project is often about starting up. New people join the project. As the code reaches maturity you release it to different teams for testing, production and the like. If there is a problem in production a developer needs to recreate the codebase as it was at the time it was released so that they can do development. Each of these steps requires that you create an appropriate workspace, build the application, and deploy and install it. Having an automated process to do this reduces the risk of miscommunication between phases and allows each phase to proceed smoothly. Consider the following scenario:

Melissa joins your team and is excited about contributing to the next version of YWA (Your Web Application). The first thing she starts to do is to set up a development environment so that she can see how the application works. Her needs seem simple; she wants to:
- Get the code for the application and place it into her workspace so that she can do development without interfering with the work of other people, especially since she is new to the codebase.

- Build and run the application. In this case the application is a web application, so there is a “deployment” phase. (She already has a local copy of the application server installed)

Melissa is aware of some of the patterns for agile Software Configuration Management, so in her mind, she assumes that she can use the Repository Pattern to help her execute a Private System Build in her Private Workspace.

She checks the code out of the version management system, runs a build script, executes a "deploy" target, and tries to run the application. Nothing works.

Ted, who sits in the next cube, explains that it "takes a while" to get the app set up, and that since he had it working, he hasn't tried to run the build and deploy script as it. Anyway, he is kind of busy fixing the many problem reports that the product has spawned, so he points Melissa to an email exchange that he had with his neighbor when he started the project. After much manual effort, Melissa can finally run the application. She prays that she will never have to install this from scratch again, since, while she thinks that she knows all of the steps, there may be a few that she forgot to write down.

Two questions come to mind:
- What is wrong with this picture?

- Why does it sound so familiar?

We can attempt to answer the first question. The answer to the second revolves around the fact that most organizations do not fully understand the costs of ignoring this problem.

The Costs of a Manual Build Process

The process that Melissa had to go through illustrates some of the problems and costs of not having an automated, simple, build process:
- It takes a lot longer than necessary for people to get started on a project. While some learning curve is always needed, that extra energy is better spent on interesting things that are not easily automated.

- This process is not repeatable, meaning that there will be more surprised than expected. You'll here the phrase "Works for me" more often than not because no everyone is working with exactly the same setup. And they may not even know what their setup is

- People will be afraid to make more than minor changes. If the system is fragile, the effort to make a minor change will be high.

- Developers will be afraid to start from a clean slate, because doing so adds significant overhead. This is basic human nature: the risk of bad feedback from taking “too long” to accomplish a task is greater than the (eventual) discovery of a problem caused by not following a good process.
The effort to simplify the “getting started” part of a development activity is small compared to the cost of allowing developers to muddle through. Most organizations ignore the costs of a broken construction process because work seems to get done in spite of it.
Consider the following situations:

- A development team can’t reproduce a problem found in production. After much manual effort they discover that the development environment is using a slightly different version of a third party library, and also a slightly different deployment configuration. The configuration files are edited by hand.

- The overhead of fixing a problem includes a day to set up the development environment, since in includes a number of manual steps.By the time you hit these cases it is often too late to fix the underlying problem, so you muddle through, and forget the problem until it happens again.

The Importance of Construction

While some recent article discuss the importance of construction [Knoernschield] , construction is an often neglected aspect of software development. While language constructs, performance, application functionality, and even version management practices grab the attention of developers, good practices in these areas are meaningless unless you can deliver a working application. To deliver a working application you need to be able to construct a working version of the complete application at every stage of the application development process; without a running application you won’t be able to test, and without tests, there is no way to verify your application’s quality.

A good build (or construction) process is essential for all types of development, agile and not, but a good construction process is a prerequisite for agile development because of agile method’s reliance on feedback; the clearest feedback is a running application along with a suite of unit tests. Without a reliable, repeatable, build process the best testing practices can be rendered futile because you have no reliable way to tell what you are building with, and what you are executing.

Fractal Builds

In most environments, construction happens at all levels of the application. There is a build that release engineers typically do. But developers must perform some sort of build process to test their code. The more similar that the various kinds of builds are, the less likely the “works for me phenomenon” will occur. This carries over to the idea that the developer’s build should be as similar as possible to the Integration Build and Release Build. While there may be subtle differences to account for environment, for example, setting up configuration files to point to a development database, it is to your advantage to have the build processes be as similar as possible so that the resulting application works the same, allowing one to both detect problems earlier (i.e., you can detect an issue in a development workspace before the application is released) and also to reproduce them more easily in the cases where there are issues discovered in production code that you did not have tests for in development.

Figure 1 illustrates the idea of a common build script. A build script along with some configuration properties should allow you to build (and perhaps deploy) for a variety of environments. A common script allows to more easily track what differs between environments.

If you include the deployment step as part of the build, your developers will be better able to reproduce problems in their workspace, and thus be able to resolve them.

Reproducible Builds

A good build process is reproducible. In 1999 Brad, Steve, and Ralph Carbrera documented some of the patterns that allow for a reproducible build [BuildPatterns]. In this article we’ll try to justify this statement. In future articles we’ll say more about how to attain that goal.

The build (and tests) determine the health of the project. And the frequency that you execute the build/test cycle determine the rate of change. Also, manual processes, no matter how well documented (and they are never documented well enough!) are not as repeatable as automated processes. This is mostly human nature; Supporting activities are often neglected in favor of more pressing concerns.

What we need is a repeatable process that allows developers to build and deploy applications in the test environment. We can make the process configurable to address local system issues, but the process should be as much like the Integration build as possible so that we can identify problems before they appear in the QA environment, and also so that a developer can reproduce problems that happen to slip through.

The key to effective build lies in the relationship between these patterns:

- Private Workspace (A workspace-side pattern)

- Repository (A Codeline pattern)

- Private System Build (A Workspace pattern)

- Integration Build (A workspace Pattern)

- Third Party Codeline We’ll discuss these patterns in more detail shortly. First, let’s examine the roles that they play in enabling an agile software team. Figure 2 illustrates the relationship between the patterns. A Private Workspace is a place where a member of a team does development.

Figure 2: Patterns

To build a Private Workspace you need to:

- Get the source code from the application. This should be a simple process and the Repository pattern describes how to do this. Since some of the components of the application are from external sources, the Repository used the Third Party Codeline pattern.

- Build the application. There are two kinds of builds that matter:

- The Private System Build, which you will use to build in the workspace.

- The Integration Build, which happens in an Integration Workspace. This build provides a definitive state of the codeline, since a Private System Build may not have integrated everything that is current.Once you build the application you will want to be able to execute the application and any tests. This may include a deploy step.

Qualities of an Agile Build Process

In his article, “Benefits of the Build” [Knoernschild], Kirk Knoernschild discusses the importance of the build process to Extreme Programming environments. Even outside of an XP environment, reliable, reproducible builds can help you to be more effective.

A developer should be able to:
- Easily create a sandbox with the appropriate artifacts for any version of the application.

- Build the application using a script (and perhaps with an IDE if appropriate)

- Where applicable, deploy and configure the application to a workspace environment.
There are a number of ways that you can achieve these goals.

- To create a workspace:

- Check everything out from a repository location.

- Run a script with a target.

- To build (and deploy) the application:

- Run a script target.That should pretty much be it. It should be a simple process. There are details in how to get there and we will examine the various Workspace patterns in detail in future articles.

(at right: Signs on the post
office bulletin board in
Childersburg, Alabama.
Photograph by Jack
Delano,May, 1941).

While automating the process steps is important, there is a certain amount of bootstrapping involved. If you can create a workspace, complete with build scripts from your version management system, you still need to tell people what to get out of the version management system. While making the wisdom of how to build the application an oral tradition has a certain amount of quaintness to it, it really isn’t an effective communications tool. One mechanism that has worked for us is to post some basic project setup information in a common place. While a project bulletin board could serve the purpose, an internal web site, or wiki works better. (And you have other information that you need to share that you would post on an intranet anyway right?) This document should contain the essential information to get started, such as:

- Where to find the installer for the version management tool client.

- Connection information for the version management server for your project.

- The locations of other tools you need to install manually, such as a build tool.

- The name of the target to run to set up the client.This should be a short document with only essential information. This document should be in an easy to find place; a URL like: intranet.company.com/ should contain a link to this document.

Who's Responsible?

You may be saying to yourself, “this sounds grand, but I’m a developer and this is a job for release engineers.” Or you may be a release engineer and think that developers needs to responsible for build scripts. While there is some merit to having different roles, we hope that you all realize that you are working on a project, and for it to succeed the goals of all people on the team need to be aligned. So, who’s responsible? You are. Yes, you, the reader of this article. If you are a developer, the build needs to work for you. The build also needs to work for release engineers. If organizational constraints divide ownership of the build scripts in a way that makes change difficult, get everyone who has a role in this in a room and figure out how to make the scripts work better for everyone. In some cases you may need support from management to bootstrap the adoption of a new process.

Even if think that you can get needed management support, you will need to put some effort into explaining the value of this process. You may need to develop a strawman build script yourself to demonstrate the difference. Perhaps you can encourage your management to create a workspace a deploy the application on their computers to understand the pain of the current process, and the value of the proposed new ones. A useful viewpoint to have is: “Do or Do Not, there is no try.” [StarWars] If you have a problem to solve, you need to decide to solve it, or decide not to solve it. Just complaining won’t make things better.

Other Issues

In this article we hoped to motivate the need for a build process that can easily get developers started. We’ll expand on these topics in future articles. Some of the issues that we still need to address are:

- The requirements of different environments (development, QA, production)

- Different types of workspaces

- The role of IDEs in the build process as compared to build scripts.

- The implications of shared libraries and components (e.g. DLLs on Windows).

- The variety and role of build tools.

- The Use of virtual machines

- The value of continuous integration and the place for build servers for distributed builds.

- The relationship between building, testing and codeline policies.

Conclusions

The Build is an essential link in the development process. Improving this link will make your life easier, whatever your role is. Take small steps towards making your process better and your work will be more enjoyable. Even if your build process seems to work, consider how you might improve it further, perhaps keeping in mind the words of Laurie Anderson: “Paradise is exactly like where you are right now, only much, much better.” [Anderson]

Resources

You can find more information on the patterns in Brad and Steve's book, Software Configuration Management Patterns: Effective Teamwork, Practical Integration published by Addison-Wesley. There is an overview, including a reference card that you can download, at http://www.scmpatterns.com. You can find more information on Kirk Knoernschild's work at http://www.extensiblejava.com.

References

- [0] With apologies to Oscar Wilde

- [wordnet] WordNet (http://wordnet.princeton.edu).

- [BuildPatterns] Software Reconstruction: Patterns for Reproducing Software Builds, by Ralph Cabrera, Brad Appleton, and Steve Berczuk. In the proceedings of the 1999 Pattern Languages of Programming Conference. http://acme.bradapp.net/repro/SoftwareReconstruction.html

- [SCM Patterns] Software Configuration Management Patterns: Effective Teamwork, Practical Integration by Steve Berczuk with Brad Appleton. Addison-Wesley, November 2002.

- [Knoernschild] Kirk Knoernschild: Benefits of the Build, Software Development Magazine, March 2005 pp 44-46.

- [StarWars] http://www.starwars.com/databank/character/yoda/

- [Anderson] “Language Is A Virus” from the Home Of The Brave.

Steve Berczuk develops software applications at Iron Mountain in Boston, MA. He has been developing object-oriented software applications since 1989, often as part of geographically distributed teams. In addition to developing software he helps teams use Software Configuration Management effectively in their development process. Steve is co-author of the book Software Configuration Management Patterns: Effective Teamwork, Practical Integration. He has an M.S. in Operations Research from Stanford University and an S.B. in Electrical Engineering from MIT. You can contact him at steve@berczuk.com.

Brad Appleton is co-author of Software Configuration Management Patterns: Effective Teamwork, Practical Integration." He has been a software developer since 1987 and has extensive experience using, developing, and supporting SCM environments for teams of all shapes and sizes. In addition to SCM, Brad is well versed in agile development, and cofounded the Chicago Agile Development and Chicago Patterns Groups. He holds an M.S. in Software Engineering and a B.S. in Computer Science and Mathematics. You can reach Brad by email at brad@bradapp.net

Robert Cowham is the founder of Vaccaperna Systems providing SCM consultancy and training to organisations. With 20 years of experience in software development, he has long had an interest in SCM, and has worked with clients around the world in this arena during the last 7 years. He is on the committee of the CM Specialist Group of the British Computer Society for whom he has organised several events, including their 2-day conference in 2003 (and their upcoming 2005 event). He has a BSc in Computer Science from Edinburgh University and is a Chartered Engineer (CEng MBCS CITP). You can contact him at rc@vaccaperna.co.uk