Bhushan Gupta

Bhushan Gupta joined Hewlett Packard, Vancouver Printer Division, as a Software Quality Engineer in 1997. He is currently a Software Process Architect for HP's Digital Publishing Services Division focusing on software process improvement and software measurements.

As a systems analyst, Bhushan worked at Consolidated Freightways from 1995 to 1997 where he was responsible for the design and implementation of a Windows-based logistics system. Prior to that Bhushan was a faculty member in the Software Engineering Department at Oregon Institute of Technology from 1985 to 1995 where he developed and taught numerous courses and coordinated the curriculum.

Bhushan has a BS degree (1975) in Metallurgy from Regional Engineering College, Warangal, India. He has a MS degree in Metallurgy (1982) and an MS degree in Computer Science (1984) from New Mexico Institute of Mining and Technology, Socorro, New Mexico. He is on the board of Pacific Northwest Software Quality Conference (PNSQC) and currently co-chairs the program committee.

Build and Deployment Process for the Web Applications

Bhushan B. Gupta
Hewlett-Packard Company

Abstract

Open source products have gained a broader acceptance for developing Web-based applications. Although the operating system, development tools and utilities, and languages are free of cost their poorly managed utilization could cost significant schedule slip and lead to project set backs. In an environment where it only takes a mouse click to upgrade your development tools and utilities, it becomes critical to have well established build and deployment processes.

This paper describes proven practices that have led to a sound and reliable build and deployment process at Hewlett-Packard. Two teams of engineers, later joined by a third, responsible for developing e-service components to build a Web application, chose to use open source development tools/utilities in the "Evolutionary Software Development Lifecycle" environment. Very early in the development, the teams established build and deployment goals with an ultimate objective of completely automated XML scripting, capable of nightly build and deployment on multiple servers. The teams established a strategy that started with a manual build and deployment and was refined in each EVO cycle to progress towards its ultimate goal. Build failures occurred often due to the use of multiple versions of tools and utilities by individual engineers. The teams enacted "an evaluate and adopt policy" for using the latest versions. The use of multiple versions, when a must, was supported by an extended directory structure. A clean deployment on a server was achieved by deleting the old code, copying the latest version from the source control system to a staging area on the server, and compiling and deploying it to a predetermined directory structure. Deployment failures, often caused due to missing or older configuration files, were overcome by deleting/updating the files via XML script. Several EVO cycles prior to the product release the team had a very reliable build and deployment system capable of nightly deployment and limited debugging. The process has significantly eased the build and deployment activity for developing future Web-based applications.

Introduction

Software build process is an activity that compiles the source code, links it with other components and produces binaries that are ready to be deployed for production. Without a successful build, software cannot go into production. The build process brings together essential component of a software product and must be in place as early as possible in the software lifecycle. In his article on "Best Practices" [1], Steve McConnell has emphasized the importance of early and frequent builds. McConnell sites minimum integration risks, reduced low quality risks, and easier defect diagnosis as some of the benefits of early and frequent builds. Michel Cusumano and Richard W. Selby [2] have described the daily builds as the sync pulse of a project.

A Web application is made up of one or more e-services and thus requires a complex build and deployment operation. The factors such as linking with the correct version of an e-service, maintaining correct configurations of development, build and production servers make this operation complex. Once in production, any versioning requires an immediate and accurate build and deployment activity placing further demands on the process.

Challenges of Open Source Environment

Developing Web applications in the open source environment poses unique requirements on a build process. Streamlining the build process is complex due to numerous open source software

tools and utilities, which are often developed and released separately. Generally, versioning and support of these tools and utilities seem to be a major problem. A typical utility is made available in the following builds:

- Release – ready for prime time
- Milestone – Unknown degree of quality
- Nightly – very unstable

Only the release version is intended for production. These versions can often be downloaded at no cost. It encourages the developers to download the latest version for production. This behavior encourages unintended use of milestone and nightly build versions. Depending upon the nature of the developers, a development environment often ends up with multiple versions. Once a developer has acquired the version she wants, it often takes a fair amount of efforts to configure the utility and put it to it's intended use. The documentation is limited and the only way to obtain some support is by using FAQ which has a long waiting period and does not match the development environment. This support will most likely not be synchronized with the needs of our development cycle, and will not be specific or in the context of our environment.

Although not specific to the open source environment, the other challenge faced by the development team is to build an application on multiple servers. The task becomes increasingly difficult if the servers have different configurations. The build and deployment process has to work on multiple platforms. Sanjay Mahapatra[3] has alluded to the benefits of platform independent builds.

Lastly, the development environment varies greatly. There are several IDEs (integrated development environments) such as Jbuilder, Visualcafe, and Netbeans, that make their way into the development environment. Although they do not cause major problems, they normally increase the complexity of the build and deployment process.

Project Information and Development Methodology

For the sake of confidentiality the project and the e-services names have been disguised. The mission of project XYZ was to build a Web application to support internet printing using three e-Services, LServe, MServe, and Nserve. These e-Services provided the underlining functionality. Each e-Service was owned by a development team and each team had a "build miester". The Mserve team was also building another Web application. LServe had a dependency on e-Service Mserve and so MServe played a leading role for LServe.

The application was based on J2EE and supported by dynamic html and JDBC. The teams were not bound by a single IDE and thus both VisualCafe and Jbuilder were being used across the board. The development was carried out in the PC environment and the deployment platform was Linux 6 with Apache Web server. The development tools included Ant (a build tool) [4], CVS (configuration management system), XML and other scripting languages. The teams used Evolutionary Development Lifecycle[5,6]. The functionality was built in small chunks with a constant evaluation either from a real or a surrogate customer.

The environment included a development, a test and a demonstration server with identical system configuration. The application was built and deployed on all three servers. The development server code was constantly changing. Building on the development server synchronized it with the other servers. In the beginning the testing server was used for both validation and demonstration. This caused lengthy validation downtimes especially when there were customer demonstrations for usability evaluations.

Build and Release Process

Setting Up Build Goals:
The build process was to meet individual developer's needs and other build miesters needs. Individual developers needed to build their components and test their code while the build meitser needed to build the system for deployment. The activity was initiated with establishing a build team. The team was comprised of two representatives from development and database teams, the system (server) administrator, and a process architect. The first task of this team was to setup build goals. Following are some of the "MUST" goals that the team decided the build process to have.

- Clean build every time
- Simple process without any need for documentation
- Build the entire system as well as any specific components
- Choose any version of a system component for a build
- Build recursively starting at any level
- Use simple tools
- Build on demand
- Build and deploy together
- Automated
- Build needed targets concurrently

Some of the "WANT" goals included:
- Be capable of driving a unit test
- Executable from a web site

Build Configuration:

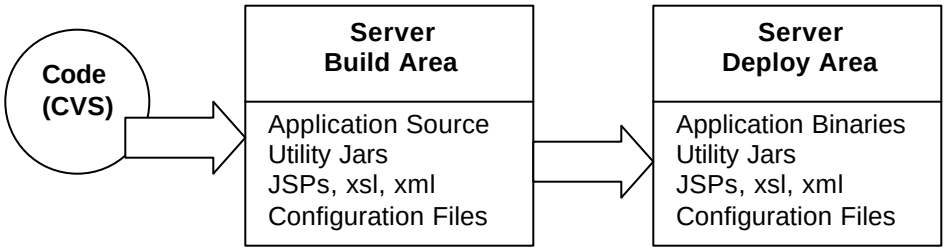

Figure 1. Build and Deployment Mechanism Layout

The entire application was under CVS, a version control utility running on a dedicated server. The code was transferred to a build area on the target server, built on the server and then deployed in a dedicated area where it could be used for production if desired.

Directory Structure:
The application directory structure was predetermined based upon the application web server. The tools and utilities were stored in dedicated directories. In the event that an e-Service depended on another e-service it was stored in the e-Service area as a jar.

Communication:
The build day and time was predetermined for the LServe, Thursday 1 PM. All the developers were reminded of the build time and asked to check in their changes to CVS by noon. Another reminder was sent at noon and code was labeled at 1 PM. Exceptions were made if a developer was working on a critical functionality. Since LServe depended on MServe, MServe was built by Thursday noon.

Incremental Approach – Stepwise Refinements:

Initially, each e-service had its own build script that was used by the build miesters. The application build and deployment was manually performed by using a Korn shell script. It was a "clean build" approach. The server was shutdown, old code was deleted from both build and deploy area. The code, including the barebones build script, was moved from the resident server to the target server using ftp. It was built in the build area and then transferred to the deployment area. Any update to the application configuration files and the server properties were performed manually. The server was restarted. The application was smoke tested and any build failures were immediately resolved. The build and deployment was then performed on the QA server. Once the failure was resolved, the system was not always clean-rebuilt immediately.

Refinement 1 – Basics :

The Lserve moved to an XML based script using Ant as the build tool. The script deleted the staging and deployment directories on the target server, ported the code over to the build area, produced the binaries and finally copied the binaries and the utilities to the deployment area. Any server configuration and properties were manually modified. The target server was "hard-coded" in the XML file and thus had to be changed for each server. The developers still manually built and deployed the new code and debugged it.

Refinement 2 – Automation:

While Mserve was still using the manual process, the LServe started to automate the build process. First the script was modified to handle any changes in the server configuration and the properties. The second enhancement to the script was to specify the server name on the command line. At this point the build process was fairly stable however, the developers were still manually building and testing their code changes.

Refinement 3 – We became diligent:

At this point NServer was actively contributing to the application. The focus shifted to more automation and providing a universal build script for the developers and the build miester. The plan to achieve it is illustrated below.

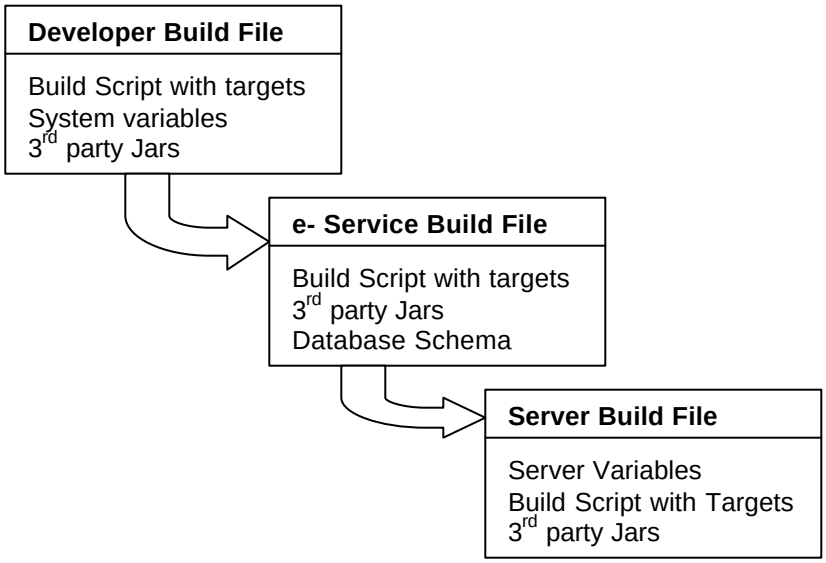

Figure 2. Final Build and Deployment Mechanism

Each developer was provided a build template which she could configure to her PC environment. Both LServe and NServe had e-Service build files and produced jar files. The final application was built and deployed by the build script on the target server.
This provided a reliable build process to build the e-Services and the applications.

Build Problems and Remedies:

Multiple 3rd Party Utility Jar Versions:
This was a major and ongoing problem. The developers freely downloaded jars often causing version inconsistency among the services and the application. There were multiple occurances of the same component
in the directory structure. This causes build problems, especially when there are multiple versions of the same component. To remedy this problem, the teams emphasized use of a single version. Domain experts were assigned to each utility and were responsible to determine the adequate version for use. The problems caused by incorrect sequencing of utility jars were corrected by proper education and communication.

System Configuration and Property Files:
The development team made changes to the development server configurations via .profile file. In the beginning deleting a property file was considered unsafe. Instead, it was decided to manually update the property files. At times these changes were not communicated to the build miester resulting in build failures. This problem was overcome by establishing better communication between the build miester and the developers and later by progressive build automation.

Human Errors and Limitations:
In rare instances the developers forgot to verify their changes. Also, some directories were moved around on the development server. The target server was not updated accordingly with the changes causing build failures. The team had very limited experience with the build tool. This was overcome by improved communication. As the build miester gained more experience, the situation improved.

Hard-coded String References:
At times the developers used hard-coded strings to specify a particular server name. Because the target server was different, it caused build failures. This was not a build problem and was overcome by modifying configuration and properties files. Several copies of the configuration were maintained, one for each deployment environment. Eventually, the strings which were moved to configuration and properties files were created as part of the build process. The build of this configuration was able to use the build parameters already a part of the progressively more automated build process.

Leading e-Service Impacted the Trailing e-Service

The service LServe depended upon MServe. MServe implemented new utilities as it was supporting another web application. The new utilities at times caused build failures.

Did we meet our goals?

We achieved most of our goals – clean build, simple tools, build on demand, build and deploy together, and automation. It was a diligent approach to develop the build and deploy process in small steps. The approach resulted in a single script for both developers and the build miesters. A big bang approach normally is not goal oriented and takes a longer time to build a satisfactory process. The process revealed that the open source environment does require more attention and communication is the key to avoiding build failures.

Acknowledgements

The author would like to acknowledge Mark Dovi, Software Development Engineer, who actively participated in the implementation of this process. The later development was carried out by Petar Obradovic, Software Development Engineer, who is still enthused about making further refinements. The author greatly appreciates their support in preparing this manuscript and accompanied presentation.

References

1. Steve McConnell, Daily Build and Smoke Test, Best Practices, IEEE Software. Vol. 13, No. 4, July 1996. http://www.construx/stevemcc/bp04.htm
2. Michael Cusumano and Richard W. Selby, Microsoft Secrets, The Free Press, 1995
3. Sanjay Mahapatra , Benefit from platform-independent builds, Java World, August 2000
4. Michael Cymerman, Automate your build process using Java and Ant, Java World, October, 2000
5. Bhushan B. Gupta and Steve Rhodes, Adopting a Lifecycle for Developing Web Based Application, Software Quality Week, San Francisco, June 2000
6. Alan MacCormack, “Product Development Practices That Work: How Internet Companies Build Software,“ MIT Sloan Management Review, Winter 2001