2007-12-04

Web Based Resource Management for Multi-Tiered Web Applications

Bryce Daniel Ott
Brigham Young University - Provo

WEB BASED RESOURCE MANAGEMENT FOR MULTI-TIERED WEB APPLICATIONS

by

Bryce D. Ott

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

School of Technology

Brigham Young University

December 2007

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Bryce D. Ott

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	Michael G. Bailey, Chair
Date	Joseph J. Ekstrom
Date	Richard G. Helps

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Bryce D. Ott in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date

Michael G. Bailey
Chair, Graduate Committee

Accepted for the School

Barry M. Lunt
Graduate Coordinator

Accepted for the College

Alan R. Parkinson
Dean, Ira A. Fulton College of
Engineering and Technology

ABSTRACT

WEB BASED RESOURCE MANAGEMENT FOR

MULTI-TIERED WEB APPLICATIONS

Bryce D. Ott

School of Technology

Master of Science

The currently emerging trend of building more complex web applications to solve increasingly more involved software problems has led to the the need for a more automated and practical means for deploying resources required by these advanced web applications. As web based applications become more complex and involve more developers, greater system redundancy, and a larger number of components, traditional means of resource deployment become painfully inadequate as they fail to scale sufficiently.

The purpose of this research is to provide evidence that a more sound and scalable test and deployment process can be employed and that many of the components of this improved process can be automated and/or delegated to various system actors to provide a more usable,

reliable, stable, and efficient deployment process. The deployable resources that have been included for their commonality in web based applications are versioned resources (both ASCII based and binary files), database resources, cron files, and scripting commands.

In order to achieve an improved test and deployment process and test its effectiveness, a web-based code deployment tool was developed and deployed in a production environment where its effects could be accurately measured. This deployment tool heavily leverages the use of Subversion to provide the management of versioned resources because of its extensive ability to manage the creation and merging of branches.

ACKNOWLEDGMENTS

I wish to thank Dave Gray, co-founder and CTO of Doba for seeing the value of this research and fully supporting and funding its execution. I also want to thank my wife Erin and my sons Will and Sawyer for their support and encouragement in finalizing this research despite my spending many late nights at the office or home in the study to see it to completion. I owe a tremendous debt of gratitude to my father for his fine example to me of education, hard work, and dedication. Finally, my thanks go out to my chair Dr. Michael Bailey for helping me schedule, organize, and overcome the various anomalies, latency, and challenges I encountered to complete this research.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 Introduction

1.1 Background

With increasingly widespread Internet usage, the emergence of more complex and reliable web-based applications has increased dramatically. With this increased complexity, the need for thorough quality assurance has also grown. These advanced web-based applications introduce a more difficult scenario for maintenance of revision information, deployment, and validation than traditional binary client-side applications.

Web applications usually rely on multiple services (which are generally distributed across multiple servers) such as dynamic and static content web servers, databases, and other web or authentication services. In order to install all required applications, there are often many complex configuration steps that span multiple servers. Any of these steps may need to undergo changes when altering or enhancing the application's functionality, and in turn must each be tested thoroughly, in particular if any of them are automated, in order to assure quality. Figure 1-1 below shows a typical multi-tiered web application environment.

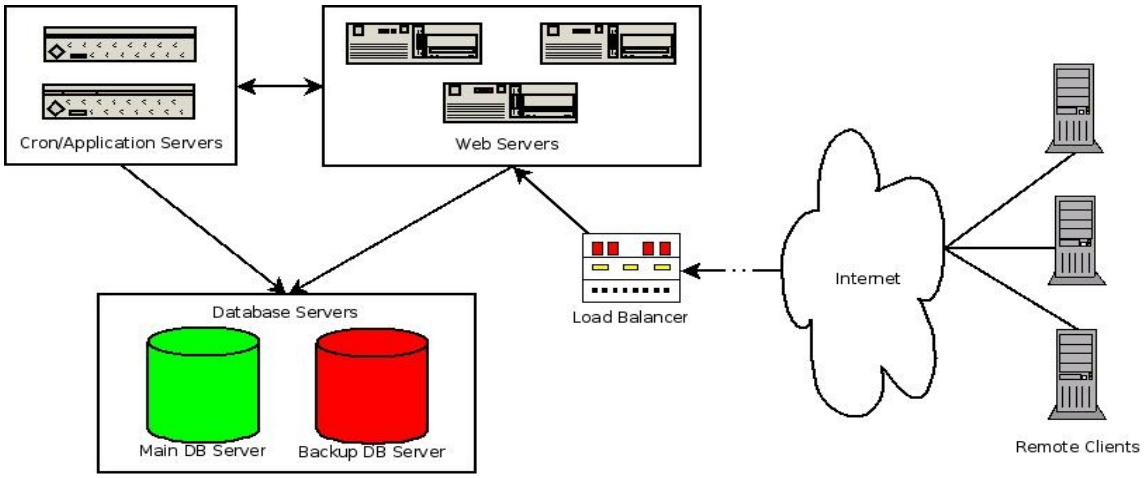

Figure 1-1 Typical Multi-Tiered Web Application Architecture

Deploying all these resources accurately becomes more important when the deployment into the production environment can have a substantial impact on application uptime and user experience. To ensure this is done smoothly, software must be deployed into the various test environments prior to its final deployment. These environments should imitate the production environment with the addition of the proposed changes against which automated and manual tests can be directed. In association with this testing, configuration and code changes to each of the affected resources must be carefully tracked to ensure that the final deployment matches the previously validated test configuration. In addition to tracking these changes, the business processes of most commercial organizations dictate the requirement of some sort of approval process for all changes or additions before they are deployed to the production environment. Traditional versioning systems such as CVS are designed for versioning source code and other binary resources, and are not well suited to the management of such complex configurations, let alone any of the

business logic involved in a modern release process such as quality assurance and management approval. With various fixes and components scheduled to be released at different times, the need for multiple test environments becomes apparent, and creating and validating these manually can become very time consuming.

The simplest and most common solution to this problem involves the use of a traditional version control system along with the manual tracking of non-source code resources. Even those resources that are versioned must be carefully managed in the various stages of test deployment. This problem is further complicated by the fact that security best practices dictate that access to servers hosting critical resources should be limited, so deployment generally becomes a manual process that must be performed by a user with administrative rights. In an environment with any more than a few developers, there will be a significant number of changes occurring to the system. The need to manually create and configure test environments, coupled with the existence of multiple servers in a distributed environment, quickly balloons resource deployment into a time intensive task, which generally must be reserved for higher paid skilled workers who can be entrusted with administrative rights on production systems.

Because of the complexity of these distributed systems, incorrect deployment, or the deployment of broken code or configurations, can require significant time and resources to properly correct or back out changes to fix the resulting error. This is partially due to the manual nature of the deployment, since it provides little intrinsic documentation outside of the versioning system, and partially due to the distributed nature of the

system, which generally requires that each resource be reverted individually. All of this has a very costly impact on the users of the system, who most certainly will be subjected to application errors or downtime while the situation is remedied, usually at a fairly significant cost to the service provider.

Outside the realm of traditional versioning software, which, as discussed, is more suited to managing source code or other binary resources, there is little in the marketplace to manage the more composite picture of multi-resource deployment. Since traditional source code versioning systems have been around for some time, they have become quite efficient, so what is lacking is something to leverage their power to manage deployment in larger distributed systems with more diverse resources by providing a scalable, automated solution to deploy resources of various types.

1.2 Problem Statement

In order to increase system quality, provide cheaper, more stable deployment, and free up valuable engineering resources for more important tasks, having a solid and scalable deployment solution is essential. While there are several mature versioning applications aptly suited for source code management, including CVS, Subversion, and Microsoft SourceSafe, none of them are able to provide deployment management in a distributed environment beyond the scope of static source code or binary resources. In addition, they do not provide a business logic level work flow process to

manage code validation and testing, and the deployment of resources to perform those functions.

Therefore, the purpose of this document is to describe the development, testing, and analysis of a system and methodology for versioning the deployment of complex web systems that include various components such as specific web server configurations, web source code, database elements, and automated scripts. The system is scalable to allow for the addition of new resources, and expandable to allow for the integration of new types of servers and services.

1.3 Hypothesis

Current resource versioning systems have attempted to manage resource versioning within the scope of their various types (ie. source code, databases, etc.), but fail to provide an overall system for versioning and deploying entire web applications. A more thorough and comprehensive system is needed to manage the versioning and deployment of advanced web applications when quality assurance validation is introduced.

The following approach is used in this thesis to create a system for managing the deployment of web based resources in a multi-stage, multi-server environment:

- A web-based application was developed for users of the system (software developers) to input their requests for resource deployment and specify where those resources should be deployed. These decisions are guided by the settings specified by an administrator of

the system which define the resources available and their associated permissions.

- The system was built using PHP and includes plug-in functionality for integration with Subversion as the primary source code versioning system and MySQL as the primary database application. It has initially been designed to interact with Linux based operating systems.
- In addition to deployment management and efficiency, the application has also been designed with information security in mind, and strives to maintain the integrity of production systems by limiting the system's access to them. With the exception of database changes, which are “pushed” to the target database, all other resources are “pulled” from their central location by each destination server, eliminating the need for the application to store authentication information for connection to each of the resources being managed.
- The deployment management system provides a business logic level work flow system to allow for the integration of Quality Assurance and managerial approval into the deployment process.
- The use of branches in Subversion has been utilized to allow for more effective management of various versions of the application being deployed. In this way, a user wishing to test or validate any version of the code simply has to check out the pertinent branch and copy the associated database.

- The system has been implemented in a real world web application environment to test and analyze its usability and effectiveness.

1.4 Justification

The manual and often tedious nature of distributed resource deployment is often a costly factor for web based application providers. As the usage of a web application increases, the distribution and virtual or geographic redundancy of its resources must also grow. The management of resource deployment to this growing architecture, including the validation of the deployed resources, without the assistance of an automated process becomes increasingly costly.

This thesis provides a robust solution for distributed resource deployment by leveraging and building upon existing advancements in versioning systems to provide a scalable and expandable system for handling the versioning and deployment of nearly any web based resource.

1.5 Assumptions

Since Subversion and MySQL are being leveraged for the management of versioned resources and database elements, it is assumed that they will perform those functions adequately. This project is not an attempt to rebuild or redesign their functionality, but instead to merely utilize it in a more automated fashion.

1.6 Delimitations

Resource deployment will be limited to those pertaining to a web application whose source code can be stored in a code repository (ASCII or binary based), and will not include operating system or other underlying software installations. For the scope of this project to be sufficiently focused, only a basic set of resources are supported which include web documents and other static resources, database resources, and cron scripts. The system has been designed so that it can easily be expanded to allow for the addition of other types of resources. In an effort to provide a cost-effective solution, preference has been given to open source solutions for the implementation of the underlying technology. As such the demonstration system was developed and tested in Linux. It is also limited, to some extent, to the types of resources it can effectively manage. The version completed for this project only supports interaction with Subversion as its source code versioning software, and MySQL as its database platform. Support for deploying cron scripts is only available for Linux based operating systems.

Conflicts have been encountered when trying to merge in changes involving symbolic links that were committed using the currently newer, revamped version of the SVN client (1.4.x) to an older version of the SVN server. Likewise, a conflict may be encountered when merging a folder that has been deleted and then re-added using the older client. Manual recommits of these resources may be required to resolve these issues. If encountered, it is recommended to update to later versions of the Subversion client and server software.

The software developed in conjunction with this research has been released into the open source community, so it has been designed without the use of any closed source or proprietary elements. As such, it may not fully support some resources that rely on integration with specific closed source software.

2 Review of Literature

This chapter describes the current solutions in the arena of web application resource management and associated technologies. It begins by discussing the current industry deployment solutions as well as their shortcomings, and moves on to describe the various types of resources that need to be managed. Each type of resource requires a different management element. These resources include static content, dynamically generated content, databases, scripted or scheduled content, and third party resources. In the context of this paper, database changes refer to updates to the database schema used by the web application for persistence, as well as the occasional change to the actual data being persisted. Next, the chapter discusses revision control and its use in managing these resources. Finally database versioning is discussed as a means for managing database resources.

2.1 Industry Survey

In order to determine what types of web-based applications are being managed in industry today, and what is involved in deploying those applications, an industry survey was conducted. Response was limited, but provided valuable insight into the state of existing solutions in the arena of web application deployment (Mohlman & Jacobs, 2007; Dickerson, 2007). A

complete list of the questions administered as part of this survey can be found in *Appendix F: Industry Survey Questions.*

From the survey, it was discovered that most commercial resource deployment is done through some sort of custom deployment process. This usually consists of a script or series of scripts that is run to deploy the specified resources into the testing or production environment. These scripts could either be some type of archiving mechanism that deploys an entire image of the production resource, or a script that simply runs a series of update commands on the target server (Mohlman & Jacobs, 2007; Dickerson, 2007). It was also found that the most common resources that are deployed are source code, static files, database changes, and scheduled processes (Mohlman & Jacobs, 2007; Dickerson, 2007).

One problem commonly encountered in the use of these methods is a sometimes time consuming deployment process that requires careful observation to ensure proper execution. In larger companies, to mitigate the risk of errors in the deployed resources, the deployment is often done in the middle of the night during off-peak hours, thereby requiring a system administrator to work odd hours. It was also noted that for some types of resources, such as SQL statements, that it was fairly tedious for the developer to define what needed to be deployed. In the case of SQL statements, deployment requests often consisted of a versioned file containing a long list of SQL that became increasingly difficult to manage as its size grew.

In response to the results of this survey, the focus of the research will be on developing, testing, and analyzing an automated and scalable solution

for deploying source code, static resources, database changes, and scheduled processes in a scalable fashion.

2.2 Web Deployed Resources

Web based services are now an essential part of the online marketplace. In many sectors such as e-commerce related services, this may be the only acceptable offering, and as such, a plethora of previously client-side or internal only services and applications have been migrated to a framework accessible from the Internet. In order for these various applications to function fully and provide a feature rich experience for the end user, they often rely on the composition of several key resources. These individual components generally consist of static scripts and resources, dynamic scripting modules, database resources, scheduled scripts or services, and third party applications or web services (Schlossnagle, 2006). All of these except for third party resources, due to their sheer number and uniqueness, will be addressed in the practical implementation of this research. Examples of these types of resources include Content Management Systems (CMS) that provide a 'what you see is what you get' (WYSIWYG) interface for updating web content, custom file upload/processing scripts, web service APIs, and web based data warehousing or application services.

2.2.1 Static Content

The two most common pieces of static content required by a multi-tiered web system are images and static HTML pages (Schlossnagle, 2006;

Mohlman & Jacobs, 2007). Most content by volume and by count is static, usually equating to more than 50% of total content being served, so efficient systems will place emphasis on properly deploying this static content. When the system has been optimized, these resources can exist in various geographical as well as logical locations, depending on the scope of the system distribution. There may also be some advanced caching techniques employed to increase distribution efficiency.

Since the scope of this thesis does not include access to online systems, instead focusing on their deployment, client side caching techniques, such as browser caching, will not be addressed here. Some pertinent server side techniques however, must be discussed. In particular, the only pertinent techniques are those that are a direct part of the resource cluster, and whose content is controlled by the resource management system. Techniques not examined include transparent and non-transparent proxy caching employed by the client's ISP, corporation, or subnet to reduce network traffic.

The first technique used to improve static content distribution is the use of specialized web server configurations optimized specifically for the serving of static content. The main trade-off with this optimization is the amount of web server system resources required by a web server compiled to serve dynamic content versus one built to serve static content. As of March 2007, Apache is the most used web server on the Internet with 58% of the 110,460, 149 sites surveyed employing it (Netcraft, 2007), so it will be used as the example for how this technique works. In order to serve dynamic content, Apache can be compiled with `mod_php` or `mod_perl`, which enables

the processing of PHP or Perl dynamically generated scripts respectively, the implication of which is a larger memory and processor footprint for each HTTP process. With a bare bones, static Apache install, one which is compiled to only serve static content, a very minimal system footprint is incurred for each thread, thus allowing a much larger number of threads to be run at once by the server. Since new threads are created to deal with increased server load, this equates to a much higher number of resource requests that can be processed by the server. Efficient systems that utilize this technique will have separate optimized web servers for serving static content from those used to serve dynamic content. From a system management perspective, this results in more targets for the management system to deploy resources to.

The next static content serving technique involves piggy backing on the first by creating a cluster of servers to handle static content requests. This allows the system to handle more requests than could be allowed by a single server as described in the first technique. In addition, utilizing distributed DNS, these servers can be distributed geographically to provide more optimized geographical service to the end clients. As long as the content being served is properly distributed to each server in the cluster (a problem to be addressed by the deployment system in this thesis), remote clients in all logical locations will view resource functionality the same way (Schlossnagle, 2006).

The final pertinent static content serving technique that could impact the effectiveness of a resource deployment system is the use of “Cache-on-Demand”, which basically consists of using a reverse proxy to help serve

static content. This has the main benefit of accomplishing the following two tasks: reducing traffic to the main site servers by serving previously viewed content from their own cache, and reducing the time spent by the main web server processing TCP connections and sending data to the clients by allowing the web server to talk over low-latency, high-throughput connections to the proxy servers. For example, an end client that is connecting via a slow dial up connection will tie up web server resources much longer than a client connecting over a speedy fiber connection, especially if the server must resend lost data. The cache server takes on the task of "spoon feeding" clients static data. The implications of this technique on a resource deployment system is that the caching context used for serving previously requested content must have a short enough life span or be expirable such that changes to the resources will appear in the cache within a reasonable period of time. Examples of software configurations that use this caching technique are Apache with `mod_proxy` and Squid (Schlossnagle, 2006).

Since the practical implementation of this research is strictly a software management solution, it will not involve the intrinsic use of any of these caching or server optimization techniques since they rely on the underlying server architecture. However, they are strongly recommended as complementary solutions for an efficient application, and are mentioned since their implementation will affect the number of separate resources and thereby efficiency of the deployment tool developed as a result of this research.

Regardless which of the aforementioned techniques is used for optimizing static content distribution, Schlossnagle (2006) recommends using some sort of revision control system, either checking out resources directly to the end server, or checking them out to a central location and then using a tool such as `rsync` to move them to each of the destination servers (to reduce the risk of long checkout times from revision control systems). This technique is heavily employed to manage content in the deployment tool developed in conjunction with this research.

2.2.2 Dynamic Scripting Content

By far, the most emphasis in web based application development has been placed on creating and optimizing dynamically generated content. While static content comprises the most accessed resources by volume, dynamic content is what provides a web based application with the most value. Whether it is a news site generating feeds tailored to the user's geographic location, or an online banking website showing the logged in user's financial data, the ability to tailor what is displayed to the needs and requests of the end user is what really gives the web its power. Imagine the time and resources necessary to maintain a static copy of each page displayed to each user on the average site from the previous two examples, and this point becomes quite apparent (Schlossnagle, 2006).

Code used to dynamically generate web content (also referred to as “server-side scripting”) can consist of code written in any programming language. While the most commonly used languages are those that have been optimized for use in creating dynamic web content such as ASP,

ASP.NET, PHP, or ColdFusion, using the MVC (Model, View, Controller) framework model, any programming language can be used to generate dynamic web pages through the creation of a web templating system (Helman, 1998). The selection of an ideal language for dynamic content generation depends on the needs of the application including speed, scalability, reliability, and maintainability. In general, languages that are pre-compiled such as C or Java will run faster since their scripts do not need to be parsed each time the page is loaded like in Perl or PHP. Many parsed languages however, also have utilities or platforms that can be used to allow them to run in a pre-compiled fashion, thereby increasing efficiency.

In the industry survey conducted of industry professionals, the types of languages used in dynamic scripting varies from C to more web-specific languages such as PHP and ASP.NET (Mohlman & Jacobs, 2007).

Since dynamic content is where most development efforts are placed, it is also usually where the most changes occur during the evolution of a web-based application. These resources are the most likely to have unexpected errors or defects and thus require rolling back, therefore in a resource management system, these are probably the most critical resources to manage effectively (Mohlman & Jacobs, 2007).

2.2.3 Persistence (Database Resources)

A critical component of most web-based software systems is persistence. Often, this function is accomplished by a relational database, which is what will be discussed here, however with a properly architected system, this persistence layer can be replaced with an ODBMS (Object

Database Management System), XML files, or any other persistence method (Mohlman & Jacobs, 2007). The management of the deployment to this type of resource is essential. In order for the dynamic scripting components to interact with the persistence layer, there must be some sort of interface, which is usually provided through an API in the specific programming language being used to dynamically script the content (Codd, 1970). This allows SQL statements or other persistence commands to be called directly from the code.

In most object-oriented languages, SQL commands are passed through a connection object which acts as the interface to the database layer. This interaction is further abstracted by the persistence architecture which organizes the code to separate business logic from that used to retrieve data from the persistence layer. This provides a logical separation of the data retrieval methods and the business logic, making the migration from one persistence method to another a seamless process as the product evolves. If the persistence methodology changes, it can be easily changed without having to touch the business logic that affects the rest of the software. An example of this type of implementation is the Java Data Objects Persistence Model which uses a persistent storage manager to handle the persistence of data (Marr, 2005).

Since the most common persistence implementation is a Relational Database Management System (RDMS), this thesis will focus on the use of RDMS resources for persistence. According to the Relational Model, data is classified according to its natural structure, without any imposed structure to allow for accessibility by the system. In essence, data can be organized,

optimized, and normalized according to its relational structure without the concern of how the system should best access it (Codd, 1970; Wikipedia, *Database*, 2007). Some of the more common RDMS platforms include MySQL, Oracle, SQLServer, Sybase, PostgreSQL, DB2, and Paradox (Bright Lemon, 2007).

2.2.4 Scheduled Scripts and Services

In most complex web-based applications, there is a need for automated services or scripts to be run as separate or background processes. These automated services may be fired off according to system triggers, or they could simply be configured to run on a set schedule. Examples of this type of service include a script that updates the status of product orders placed by users when information from the shipper is input into the system. If the shipping info is input via an API or other system call, this action should be performed independent of a user clicking a button on some specific web page. One of the more common forms of scheduled events in a web-based application is the utilization of a cron job in Linux. These can be configured to run applications or scripts in recurring fashion on a set schedule, and therefore will be addressed by the research in this thesis.

Figure 2-1 shows an example of a cron file entry as it would appear as part of a cron script. This particular implementation is created for the `build` user and will change to the user's home directory and run the PHP script `process_order_updates.php` as the `build` user every hour on the 12th minute. Similar cron entries can be utilized to automate the execution of any scheduled component in a web-based system (Little Tech Shoppe, 1994).

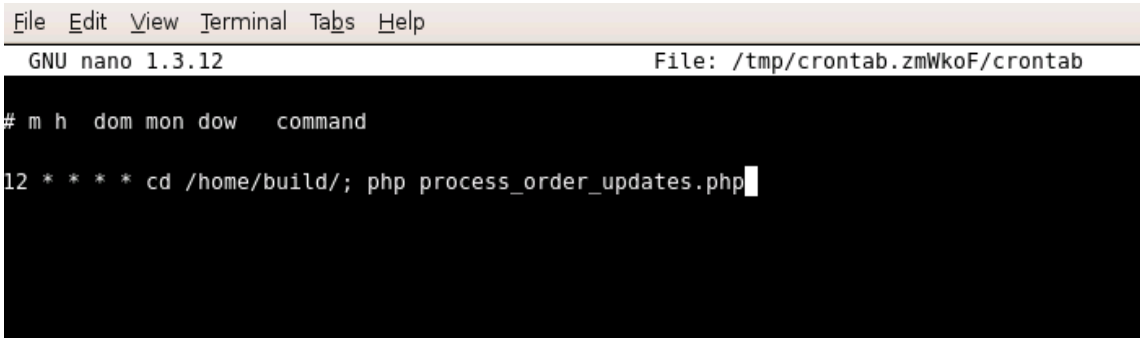

Figure 2-1 Cron File Entry

2.2.5 Third Party Resources

In addition to custom web scripting components, the final piece of web-based applications worth mentioning is the use of third party resources. These can consist of code modules, remote API calls, web services such as SOAP or RMI, or even entire applications such as live chat or content management systems (Mohlman & Jacobs, 2007). For the scope of this thesis, only those third party components that can also be managed by a database or code versioning system will be addressed.

2.3 Revision Control

Revision control processes were actually spawned from a different discipline than software engineering or computer science. The processes began as a means of tracking versions of engineering blueprints so that when a design forked and one path was chosen and it later turned out to fail, the design could be reverted back to the point of the split (Wikipedia, *Revision Control*, 2007). As software development methodologies progressed, it became apparent that the revision control process could be of great benefit to that discipline as well. Revision control has since evolved

into a process for the sharing of electronic data as well as any other medium that is to be accessed and edited by a group. Most modern software development processes recognize revision control as a necessary component to success, and it will be heavily relied on in this thesis as a solution to provide resource management.

2.3.1 Revision Control Defined

Versioning in software engineering has been classified as consisting of three main concepts. These are software *modules*. *object-orientation*, and *components* (Speck & Pulvermuller, 2001). In 1968 at the NATO conference in Garmisch-Partenkirchen, the research area of software engineering was first brought up, relating it to other engineering disciplines such as mechanical and electrical engineering (McIlroy, 1976). One of the first approaches in this new discipline addressed the need to divide increasingly complex systems into smaller software *modules* so that they could be more easily managed (Parnas, 1972). The concept further evolved to include the separation of logic into functions or procedures, and was dubbed procedural software development, thereby solving one of the major problems of software engineering; the separation of data and operations that function on that data (Speck & Pulvermuller, 2001).

The concept of *object-orientation* enhances the benefits of using modules by providing objects that can be used to encapsulate both data and functions. This approach helps solve the problem of visibility, creating “public” and “private” methods and attributes, which can be used to encapsulate data and functions that only belong to the object versus those

that can be accessed outside it. The idea of classes was introduced to help group together objects with similar functionality. Relationships within this realm include association, aggregation, and inheritance. This methodology of *object-oriented* design and its awareness in software development has led to one of the most powerful means of software development: the use of patterns (Speck & Pulvermuller, 2001). Introduced to the software development process by Christopher Alexander, an architect and civil engineer, patterns became a very efficient way to reuse software components (Alexander, 1999). This use of patterns eventually led to the development of frameworks which contain well defined yet flexible elements that can be used to customize software to the needs of the application (Fayad & Schmidt, 1997).

The final piece in the basis for versioning is the concept of *components*. Since it has been proven that frameworks are not always flexible enough, the idea of *components* has been propagated. A component consists of one or more objects flexibly composed (Szyperski, 1997), and these flexible elements can be used to create larger components or applications. Examples of component environments include COM and CORBA (Speck & Pulvermuller, 2001).

While the aforementioned concepts create the basis for versioning, they fail to provide solutions to the problems caused by the existence of multiple systems created from a common base. To address this, there must be a methodology to manage the variability of components within these composite systems. This is the end goal of revision control. A revision within version control can be defined as a set of conditions of a particular version

as well as the conditions of any sub-versions that exist with that version (Speck & Pulvermuller, 2001). An optimized revision control system will capitalize on this definition, using existing versions in the definitions of new ones, thereby creating normalized results.

2.3.2 Implementations

Since the concept of revision control has been around for some time, there is an abundance of systems that have been built to version software components. These systems have steadily evolved, and currently there are some very effective technologies being used that help solve the problems mentioned that result from complex systems of composite components. This section will discuss the most pertinent of these implementations.

2.3.2.1 Concurrent Versioning System (CVS)

One of the most popular revision control platforms in use today is the Concurrent Versioning System also known as CVS. Originally invented and developed in the 1980's by Dick Grune, CVS is licensed under the open-source GNU General Public License, and is available on nearly all major OS platforms (Grune, 2007). CVS works in a client-server model which allows multi-user collaboration, even in a geographically remote setting. The software can be configured to allow networked access over the Internet to serve versioned content to anywhere in the world, even over slower dial-up connections.

CVS also employs the concept of *unreserved checkouts*, which allows more than one developer to work on the same file at one time. This is

contrary to the methodology of earlier, more primitive versioning systems that placed locks on a file being edited that had to be freed before another user could access the file. This functionality is accomplished by proving a check-out/check-in process that allow users to check out working copies of the target resources, apply changes or additions, and then commit the resources back into the system, where other users can then access them. Each version of each resource committed is assigned a revision number that can be used by the system to identify it from other versions of the file (NonGNU, 2006).

Another very beneficial feature offered by CVS is the ability to create branches and tags. This is essentially a way to assign a name to a group of revisions, or versions of files. This can be done to create a release version of the application, or it could be used to create a side branch of the software that can be utilized to develop and integrate large changes before merging it back into the main version. This ability to branch and tag is essential for development environments that support a large number of developers and frequent code releases or changes. Other useful features offered by CVS include the ability to run scripts to manage check-in and check-out practices, robust revision history browsing, and file conflict resolution (Online CVS Manual, 2006).

2.3.2.2 Web Distributed Authoring and Versioning (WebDAV)

From the early days of the web and the inception of HTTP, its creators had in mind the idea to support versioning over the web. However, originally there was no mechanism like resource locking in place in the protocol to

allow this to occur. This void has been filled by Distributed Authoring and Versioning Over the Web also known as WebDAV. WebDAV extends the HTTP protocol by defining a new set of HTTP methods, headers, and status codes. XML is utilized instead of message headers to send structured data in the message body of an HTTP transaction, utilizing the existing structural functionality available in the XML standard. In addition, unlike message headers, XML has no size constraints, allowing it to be utilized to define an unlimited amount of data (Hunt & Reuter, 2001). These extensions to the HTTP protocol provide six additional functionalities: overwrite prevention, properties, name-space management, version management, advanced collections, and access control (Whitehead & Wiggins, 1998).

2.3.2.2.1 Overwrite Prevention

Probably the most important mechanism for revision control that WebDAV provides is the overwrite prevention mechanism. If two or more people are allowed to write to the same unversioned document, valuable changes can be lost as first one person writes the resource and then another replaces it without first merging in their changes. To address this issue, WebDAV provides an *exclusive write lock* that ensures only the owner of that lock can overwrite the resource, and a *shared write lock*, which allows a collaboration of contributors to work together on a resource. Release of locks occurs with an automatic timeout, thereby reducing the lock administration costs (Whitehead & Wiggins, 1998).

The scope of WebDAV locks can be on a single resource or on a hierarchy of resources. In order to determine if a lock exists on a resource,

there is a lock discovery mechanism that utilizes the property infrastructure to tell authors whether there is currently a lock on a particular resource. By its nature, the web is always readable, so there is no concept of a read lock in the system. One important implication of this is that in a “writable” web environment, if a lock is not owned by the author, contents of the resource may change without warning. **Figure 2-2** below describes the protocol implementation for a generic WebDAV client (Whitehead & Wiggins, 1998).

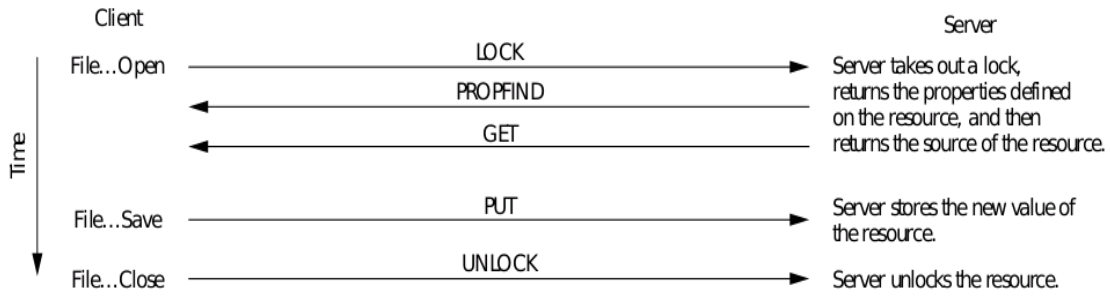

Figure 2-2 WebDAV Protocol Implementation

2.3.2.2.2 Properties

Another useful mechanism provided by WebDAV is the ability to assign metadata to versioned resources through the use of properties. This metadata can provide the important function of making resources much more searchable, by storing additional information about them besides their content. These property values are expressed using XML in a name, value pair structure, and can be assigned to any versioned resource. Using XML for the storage mechanism of this valuable metadata has additional benefits as well. Since XML mandates support for UTF-8 and UTF-16 encodings, it has built in internationalization, allowing WebDAV properties to contain

virtually any language characters. The use of XML also allows WebDAV properties to support integration with other metadata activities that utilize XML, such as the Resource Description Framework (RDF), and the Dublin Core, which is a schema for the creation of digital library catalogs (Whitehead & Wiggins, 1998).

2.3.2.2.3 Name-Space Management

In order to support a distributed system of web authoring, users must be able to enumerate what resources currently populate the name space, as well as be free to copy, move, and delete them should they desire. WebDav provides part of this functionality through the use of collections, by providing direct management of local resources, and referential management of remote ones. *Copy* functionality in WebDAV affords the author the ability to modify and backup resources, as well as enact changes in ownership. The ability to *rename* allows the changing of naming conventions or the correction of typing errors.

2.3.2.2.4 Version Management

Version management in WebDAV allows for the tagging and storage of important document revisions for future retrieval. Similar to the branching mechanism discussed earlier in CVS, it also allows multiple developers to work on the same document in parallel. The automatic versioning mechanism does not require clients to be version-aware. Instead, the server handles all necessary functionality to record all modifications made to

resources. Specific revisions can be accessed or tagged as desired to provide desired compositions of resources (Whitehead & Wiggins, 1998).

2.3.2.2.5 Advanced Collections

Directly analogous to directories in a file system, advanced collections, as introduced by WebDAV, allow the representation and managing of groups of resources (Hunt & Reuter, 2001). These groups, which contain referential members, provide a mechanism for hierarchically organizing managed resources. Referential resources act like symbolic links in a file system, allowing resources to be reused in various places without having to duplicate themselves. In addition, this allows the collection to contain non-HTTP resources. Also supported is the use of ordered collections that can be arranged in a client-specific fashion, creating more useful human-ordered resources, such as the chapters in a book (Whitehead & Wiggins, 1998).

2.3.2.2.6 Access Control

In an environment where remote write privileges are granted to clients, it is essential to have some sort of access control mechanism in place. WebDAV supports this through the use of HTTP Digest Authentication, the cryptographic authentication scheme introduced in the HTTP 1.1 protocol (Whitehead & Wiggins, 1998). This authentication scheme uses multiple one-way hashes to encrypt the user name and password pair, providing a proven and much more secure method of

authentication than most remote authoring tools (Whitehead & Goland, 2007).

2.3.2.3 Subversion (SVN)

The latest, and increasingly most popular solution to revision control is a piece of software called Subversion (SVN for short). Built as the next-generation solution to resolve some of the lingering issues in CVS, it is built on top of the WebDAV standard, expanding on its functionality to provide a more thorough and featureful revision control solution. Since it is meant to be an improvement on CVS, SVN essentially provides the best combination of these two technologies.

The list of improvements in SVN over CVS is fairly extensive. First of all, directories, renames, and file meta-data are all versioned. Utilizing the properties functionality from WebDAV, SVN allows the user to define custom properties on any versioned resource. Included in this functionality are the definitions of some reserved SVN properties, the most notable of these being *svn:externals*, which allows a separate SVN resource (usually a folder) to be linked to another, such that when the linking resource is checked out, it will retrieve the linked resource. This acts like a symbolic link of sorts. Another notable change from CVS is the fact that commits are truly atomic. No part of the commit is implemented until the entire commit is completed. This prevents partial commits in the event of network failure or other errors during commit, which can cause significant version corruption. This is partly achieved by the fact that in SVN, revision numbers are assigned to entire commits rather than individual resources. This also makes browsing the

versioning logs simpler, since a single version number can be used to determine what resources were committed as part of the change.

Since SVN is built on top of WebDAV, it is designed to perform well in the HTTP environment. As such, it has been thoroughly integrated with Apache, allowing it to utilize the functionality of many Apache modules including its access control and content serving mechanisms. If Apache is not the desired mechanism, the SVN server will also function fully as a standalone server module.

When creating branches or tags of resources, SVN takes advantage of referential relationships to create what equates to symbolic links to the actual resources. This equates to very quick constant-time operations when performing these functions, and to optimized usage of persistence resources since the data only exists in a single place inside the SVN persistence. Additionally, the costs of alterations are proportional to the changes being made, not to the size of the resources being changed, since only the changes are persisted. This is equivalent to taking an incremental snapshot and only storing the changes.

Written from the ground up as a client/server application, SVN's functionality is very modular and has well defined interfaces, eliminating many of the integration problems that have plagued CVS in the past such as insecure remote access, non-atomic check-ins, unversioned directories and meta-data, unversioned symbolic links, and time consuming branching and tagging. Subversion allows the versioning of symbolic links, which is not supported in CVS, and provides efficient handling of binary files. Other notable features of SVN include parseable message output, localized

messaging based on current locale settings, and the ability to mirror a repository for redundancy (Subversion, 2007; Subversion Book, 2007).

As a result of these desirable attributes, Subversion was chosen as the version control system for use in this thesis. While Subversion provides exceptional support for ASCII and binary based files, its inability to handle the execution of database resources makes it an incomplete solution by itself for this research. It also lacks the ability to provide the injection of QA process and other business logic into the release process.

2.4 Database Versioning

One of the more difficult and complex problems in software versioning is that of versioning database elements which can include both the schema and the data itself. During the lifespan of an application, the database schema will usually evolve, undergoing several iterations. These changes in the schema can in turn cause changes in the data stored in the system, and are usually also tied to changes in code that utilizes the database for persistence. In order for these changes to be effectively versioned the state of both the schema and the data must be stored. On top of an evolving schema, data in a web-based application is constantly being inserted, updated, and deleted.

The need for maintaining data under a schema definition which undergoes alterations and evolution is not a new issue (Roddick, 1996). All of these factors combine to create a very compelling

argument for the use of database versioning in advanced web systems. The two most common techniques for database versioning are *schema evolution* and *schema versioning* (Wei & Elmasri, Study and Comparison, 1999).

Schema evolution in essence, involves storing only the current version of the schema and related data. This may also be referred to as a snapshot database since only the latest snapshot of data is preserved. When a schema changes the data must be altered to match the new schema, which may result in data inconsistencies. In addition, any legacy part of the application depending on the old schema must also be altered to match the new schema. This is by far the most rigid and least complex method of database versioning (Wei & Elmasri, 1999; Wei & Elmasri, 2000).

Schema versioning is a technique involving the creation of new schema versions for each evolutionary change, and converts the corresponding data while still ensuring the preservation of old schema versions and data. This allows the introduction of a temporal element to the versioning of database elements. In its most basic form, there are limitations to this method, since while it versions all schema and data changes, it only allows modifications to be performed on the most recent *version. Bi-temporal schema versioning* resolves this issue by allowing the system to perform proactive and retroactive revision changes. This flexibility comes at a cost and can become fairly complex as the past and future versions must be considered when making changes (Wei & Elmasri, 1999; Wei & Elmasri, 2000; Moreira & Edelweiss, 1999).

Because of the complexity of database versioning, its reliance on the underlying RDMS software architecture, and the constraints of the available database resources for use in this research, it is not addressed, but is strongly recommended as an area of future research. Unless a custom database versioning solution is created, it relies intrinsically on the underlying database architecture. It is presumed therefore that database versioning could be added by simply adding support for running SQL queries through such an architecture, whose SQL format should closely match the standard one employed in the research.

2.5 Existing Enterprise Systems

While conducting this research, it was discovered that there are several enterprise systems that employ some of the items discussed in this system for solving the deployment management issue. This section describes some of their implementations as well as benefits and shortcomings.

2.5.1 Enterprise Java

As part of the Enterprise Java (J2EE) implementation, packaging and deployment has been thoroughly addressed. First of all, there are several different Java runtime containers. The first of note is the Application Client Container, which handles stand alone client applications, but still allows them to interact with the application server. There is also a Web Container, which provides interception for requests sent over HTTP, FTP, SMTP and other protocols. Finally, there is the EJB (Enterprise Java Bean) Container, which provides containment and request level interception for business

logic. It also offers access for EJBs to several enterprise level Java services and interfaces like those to handle database interaction, remote procedure calls, and email functionality. The use of this container functionality in the context of web applications allows the applications to utilize standard functionality provided by an application server such as authentication, load balancing, fail-over routines, transactions, and access to server side variables, providing a much simpler and object-oriented organization of resource access and management.

Independent of the container that it runs within, there is also packaging that occurs to manage the organization and deployment of Java applications on an application server. In the instance of a web application, this is a WAR (Web Application Archive) file, which can contain any number of code modules that add business logic functionality to the web application, as well as files that describe the GUI portion of the web application. This WAR file is then added to an EAR (Enterprise Application Archive) file that may also contain other resources such as EJBs and other WAR files. This EAR file is then deployed to the Java application server and its individual components are deployed to their respective containers (J2EE Packaging and Development, 2007).

Versioning in this scenario is handled by a manifest file contained within each EJB, JAR, or WAR file that indicates the version of the resources it contains. In this way, versions of web application resources can be managed, even dynamically as they are deployed to the web server (Co-Hosting Multiple Versions of J2EE Application, 2007).

It should be apparent from the descriptions provided above, that deployment in the J2EE world has been well thought out, however, it is not ideal for addressing the issues presented in this thesis. First of all, it is confined to only managing source code. It does not address the deployment of other web application resources that have been discussed, such as databases and cron files. Also, the source code it can manage is exclusively limited to Java, eliminating any of the other languages that have been identified as common in the realm of web applications. For these reasons, the Enterprise Java model is not the ideal solution for the applications addressed in this solution.

2.5.2 Parasoft WebServices Solution

Another example of a commercial deployment solution is the WebServices Solutions from Parasoft, an enterprise software development consulting company (Web Services Solutions, 2007). The WebServices Solutions target the prevention of errors in the software development and deployment process. In order to accomplish error prevention and monitoring, a custom transparent layer is integrated into key development processes. This allows the system to utilize source control, nightly build systems, and other error prevention techniques that include adherence to coding standards, and unit and regression testing to prevent development and testing errors.

The Parasoft solution builds its functionality around three defined roles in the software development process; architects, developers, and product managers. Implementation of the system involves five steps:

evaluation, customization, automation, training, reports. Each of these steps is done through specialized consulting from Parasoft staff, and is meant to enhance the existing development process of the target institution (Java Solutions, 2007).

Although based on sound principles of development improvement practices and targeted at several different programming languages, because of the custom, consulting nature of the Parasoft WebServices Solutions, it is apparent that this not a feasible solution for many entities seeking to create a solid deployment process. Rather than providing a software application for managing web based resource deployment, it provides consulting services for improving the efficiency of an existing software development and deployment process. This makes it unsuitable for satisfying a solution to the problem presented in this thesis.

2.5.3 Microsoft Office SharePoint Server 2007

A commercial platform solution provided by Microsoft, Microsoft Office SharePoint Server 2007 integrates more business centric processes such as the management of intranet and organizational resources with IT related server administration, and application extensibility and interoperability.

Business processes and collaboration in Microsoft Office SharePoint Server 2007 are faciliated through centralized access to store and search content, concentrating on the control of storage, security, distribution, reuse, and management of web pages, PDF files, email messages, and other

collaborative content. This allows for the central implementation of data reporting, and tracking as well as the prevention of redundant activities.

IT configuration and management processes are facilitated in Microsoft Office SharePoint Server 2007 through the use of a single consistent administrative interface to manage internal and external facing resources as well as employ APIs and XML web services to enhance the integrated services that are offered. Some of the services offered include wikis and blogs, document collaboration, RSS feeds, discussion boards, project and task management, contacts and calendars, email integration, and integration with other Office 2007 applications (Microsoft Office Sharepoint, 2007).

As mentioned, Microsoft Office SharePoing Server 2007 provides an immense level of enterprise deployment and content management, but it has several significant drawbacks. First of all, as is the case with many commercial solutions, it is tied very tightly to the proprietary formats, and is limited to dealing easily with other Microsoft software. This makes it difficult to utilize to manage solutions build on open source or other proprietary software. Its immensity and complexity also makes it very unattractive for smaller, simpler applications that don't require all the “bells and whistles” offered in Microsoft SharePoint Server 2007. These issues make it a poor solution to the problem presented.

2.5.4 Vignette Content Management

There is an immense number of Content Management Systems (CMS), which all have very similar feature sets. Their purpose is to provide a simple

non-programmatic way for users to quickly change and publish web content. One such example is Vignette Content Management.

Vignette Content Management is a commercial product (there are also many similar open source options available) that allows access to manage users and their authorization to alter and access web based content and collaborate on tasks utilizing email, desktop applications, and a web based workspace. Pre-built templates and content types allow for easier content creation. It also provides a virtual repository to allow for the storage of content in virtually any format including database, XML, rich media, images, and flat file resources. Since the system is standards based (J2EE, .NET, XML, and web services), it can integrate well with existing services and solutions that are also standards compliant (Vignette Content Management, 2007).

While Vignette Content Management and other CMS systems are ideal for managing web content generated by non-technical users, they fail to adequately provide a solution for managing the types of resources common to a web based application. They are geared more towards managing media, text, and other “static” content, and do not generally provide versioning for source code, database resources, or other dynamic web application resources. They also usually do not offer multi-server deployment, and for these reasons are not the proper solution for addressing the problem of this thesis.

2.6 Previous Deployment Process

As will be described in more detail in later chapters, the software developed in conjunction with this research will be tested and validated within a real-world scenario to deploy resources for the primary web application of Doba, an online product sourcing company. In order to demonstrate the need for an improved system of deployment, the previous Doba deployment process will be discussed here.

2.6.1 Process Description

Previously there were several types of resources that had to be managed as part of the Doba web application. These included static web content such as images, PDF and multimedia files, and HTML pages, as well as the source code for dynamic web scripting content. In addition, there were also database resources that had to be managed, scheduled processes that ran periodically, and third party resources that included a CMS (Content Management System) and a hosted FAQ (Frequently Asked Questions), chat, and a customer interaction application. Requests for resource deployment within the Doba environment were often referred to as “pushes” or “push requests”. The terms “push” and “deploy” are used synonymously in this document.

Previously, the deployment environment for the Doba web application consisted of a principal development environment where engineers developed and tested changes to the application, a staging environment that was used to test the deployment of a set of resources as well as its interaction with other changes, and finally, the production systems.

Subversion was used to manage ASCII and binary resources, and each environment had an associated version of the required database resources, as well as a working copy instance of the pertinent source code. All versioned source code was committed to and checked out from a single trunk location for each project, regardless of its environment, and SQL changes were simply executed in their respective environments. Scheduled processes were manually run within the two testing environments and consisted of scheduled cron scripts within the production environment.

Within this process, developers' code and test resources resided within their personal development sandbox until they were ready for promotion to the staging environment. At this point, the developer would commit the versioned resources to the trunk of the Subversion project, and send an email to the deployment manager containing a list of the committed files as well as a list of any SQL changes or manual scripts that needed to be run. The deployment manager would then log onto the staging server and run an *svn update* on the list of files that were specified by the developer, thereby updating them to the latest revision within the working copy. The manager would also execute any SQL statements and process any scripts that were requested. The developer had to then validate that his changes are working correctly in the staging environment and either repeat the process to fix any defects or missing resources, or repeat the process with a new email to promote the resources to the next step which is the production environment.

On a specified deployment day, the manager would compile a list of all resources to be updated for the specified environment. This would consist of

a list of files to be updated, SQL statements to be run, and scripts that needed to be executed. He would proceed to update the working copy of the target environment with all of the changes, and upon completion, send an email to all submitters with the output results of the deployment. This process is fairly typical of what occurs with other players within the web application software industry (Mohlman & Jacobs, 2007; Dickerson, 2007).

2.6.2 Process Deficiencies

There were several deficiencies present within the previous Doba deployment process. The first of these, is the fact that there was no formal record of what has been deployed. The only significant record was the email archive that was kept by the deployment manager, which contained the request emails he received as well as his responses of what was deployed. In this format, the results were not very searchable and did not always thoroughly document who was responsible for the resources that were deployed.

The next major defect with the previous release process was the fact that the SVN changes in each environment were simply updated from the trunk into a local working copy. The first major implication of this method was that the latest revision was always deployed. Therefore, if one developer committed a change, submitted a request for deployment, and another developer committed a change in the same file before the first change was deployed, the second change would also inadvertently be deployed. Furthering the danger of this methodology is the fact that since the update was done on a list of files, over time there developed within the working

copy a complex mix of revisions that was impossible to duplicate or roll back should the need arise.

The next problem stemmed from the fact that the Doba production environment consists of several resources that contain redundancy, particularly resources housed on the web servers. In order to handle the load seen on the application, load balancing is used to divide traffic between several web servers. This also means that the same SVN resources must be deployed to several different servers. In order to simplify this, a bash script was written, but it had to make an SSH connection to each target server and execute the *svn update* command on it. This required the public key ssh credentials to be stored on the server running the script, which created a security concern since a breach of this server would result in a breach of all others.

Several issues with the previous deployment process had to do with the communication medium uses to request deployment. Since the method for communicating which resources should be deployed was through email, several associated problems arose. First, during the cutting and pasting of resources from the email to the list used by the deployment manager to process requests, cut-and-paste errors sometimes occurred, resulting in missing characters or lines. The use of email clients also had implications, as many of them would insert line breaks into long lines with a lot of text. This was especially damaging to SQL statements that were inadvertently injected with new line characters in the middle of a long string field. Additionally, since a separate email was sent for the staging and production pushes, the resources that were deployed and tested in the staging environment may not

actually be the same ones that were deployed to production. If the user added, altered, or forgot to include a resource that was deployed to the staging environment when sending his production request, unforeseen errors could occur on production.

Another issue with the previous release process was that there was no way to inject any type of business logic or document that proper validation occurred. Since developers simply sent an email with the resources they wanted deployed, there was no continuity between what was deployed to staging and what was requested for deployment to production. There was no formal way for there to be quality assurance or project management sign-off on what was being requested for deployment.

Since there is an abundance of automated scripts run through cron jobs in the Doba system, this is an important resource to manage. With the old system, this was handled entirely manually by the push manager, who would remotely log onto the target machine and manually update the specified cron file. This file was not versioned, so there really was no log of historic changes made to it. In contrast the new deployment tool forces cron files to be versioned using SVN thus they can be edited and versioned just like any other SVN resource.

Finally, since a significant amount of the previous process was manual, it was also fairly time consuming and required a trusted and skilled deployment manager with administrative rights on the production systems. He had to spend his time cutting, pasting, and cleaning up resource definitions, processing resources, ensuring correct execution, and emailing responses for each deployment.

2.7 Review of Literature Conclusions

Advanced, multi-tiered web applications contain various resource components that can benefit greatly from versioning and an intelligent system architecture. These include static scripts and resources, dynamic scripting modules, database resources, scheduled scripts or services, and third party applications or web services.

The versioning of static resources, dynamic code components, scheduled scripts, and to an extent third party applications, can be managed effectively through the use of version management software designed for ASCII and binary files. The latest iterations of open source software in this realm is quite advanced and nearly all issues that are encountered when versioning these types of resources have been resolved due to the large amount of research and development already devoted to this software. As such, leveraging the benefits of these existing systems will be the wisest course of action for creating an automated deployment system for web based applications. Because of its limitations in providing support for binary content, the versioning of directories, and the minimal support for effectively merging and branching, CVS is not recommended for the versioning solution in this research. Instead Subversion has been selected and will be implemented as the core piece for this functionality.

In order to effectively version database resources in a web based application, it is also necessary to utilize an existing application that effectively handles schema versioning. It is preferred that this also includes bi-temporal versioning if performance requirements permit it. At the very least, the platform upon which database resources are built should support

database evolution to allow the alteration of the system as it evolves. Since the scope of this research does not allow for the use of a full-fledged database versioning solution, it will instead store a simple log of all SQL statements that are executed against the managed resources.

3 Research Procedures

This chapter describes the design of a web-based resource deployment tool system and the various components of its configuration. This system is used to automate the deployment and management of resources in a multi-tiered web environment. It is designed to create a system that enforces business logic for the release process as well as provides the technical framework necessary for automating resource deployment to both testing and production environments.

3.1 System Overview

This research involves the creation of an automated tool to deploy code, database, and cron changes from the main development trunk of a versioning system to testing and production servers. The deployment of a group of defined resources, called a “push,” when done to testing servers, by default does not require administrative approval (although this can be configured so newer or probationary developers can be more closely monitored), except for those containing script requests, since this type of resource poses a security risk. Pushes destined for production systems require an administrator to review that they have been validated before actually deploying them to the live servers.

The system also has emailing and message sending capabilities integrated into it to communicate with other users and allow a user to be

notified when certain events have been completed or require their attention, such as when a push request is processed.

A version branch is created both for testing and production versions so that code can be committed and checked out from any of those branches, facilitating the ease of code management and checkout for specific server types (testing or production).

Figure 3-1 below describes the core functionality of the deployment tool implemented during this research. Developers commit code to the SVN trunk and submit a "push request," which is a formal specification of which resources are to be deployed. When push request is processed, the specified file revisions are then merged into the testing branch which was previously created through the tool as a copy of the production branch. The entire contents of this testing branch can then be merged into the production branch to complete the deployment process.

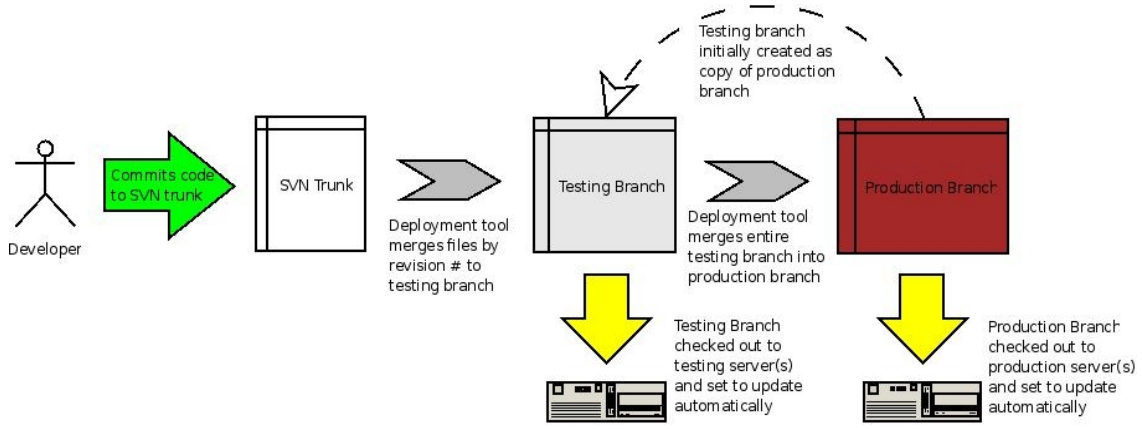

Figure 3-1 Deployment Tool System Overview

The following sections describe the functions available to each type of user in the system. Further on in this chapter user case diagrams are given for each user type.

3.1.1 Feature List

The following is a list of features currently employed by the code deployment tool that was implemented for this research.

3.1.1.1 System

The deployment tool allows the management of resources hosted on any Linux based OS. The resources that can be managed include:

- Resources managed by the Subversion (SVN) versioning system (includes ASCII and binary resources)
- MySQL databases
- Cron file configurations
- Any command or script that can be executed from the command line

The system also does the following:

- SVN resources are pushed by revision number. By default, only changes made in the specified revision will be pushed. This prevents changes committed in a previous revision of the same file from unintentionally being included in the push. There is also an option to push all changes made up to the specified revision so as to include any previously committed changes.

- Implement a QA approval process that can be used to give approval or disapproval on whether a particular version of a feature or the entire application is ready to be released
- Integrate @Task (online project management) (@Task Project Management, 2007) software to view the project management resources for a particular push request
- Manage application versions using branches in Subversion
- Allow testing branch creation from a project trunk or a production branch
- Exact changes pushed to a testing environment will also be pushed to the corresponding production environment, facilitating more accurate testing scenarios
- Offers support for SVN externals to allow external, static projects to be linked into another SVN project. This requires some special handling to be accommodated, but it is managed by the deployment tool.
- Allows the specification of a list of email addresses to be notified when a branch push is successful and also when it fails. This allows external groups to be notified when production level changes occur through the deployment tool.
- Testing databases can be created from uploaded schema files or versioned schema files stored in SVN
- Testing branches can be created automatically on a specified schedule

- Pushes made are not actually employed on the target system until all changes have been made, including dependencies that may exist between branches (like those seen when pushing SVN externals that exist in a separate branch)

3.1.1.2 Security

Having been built from the ground up with security in mind, the deployment tool has the following security features:

- Recommended and default configuration with SSL access
- Support for SSL and other secure access methods to Subversion
- Does not centrally store authentication information for managed resources, except databases, but database access can be limited to only allow required permissions
- Resource access permissions can be set on the user or user group level
- Brute force login cracking protection. Accounts are locked out after too many failed login attempts.
- SQL injection prevention is built into all core objects
- Recommended and default protection to safeguard SVN resources (branch level authentication). Authorized read only access is utilized on all automated branches created by the deployment tool. This prevents unauthorized access as well as disallowing manual writes to those automated branches that

usually lead to conflicts when merging or processing the branch.

3.1.1.3 Push User

Push users are users of the deployment tool that are able to enter, edit, and review requests for the deployment of specific resources. These are generally software developers, and are able to do the following:

- Create push requests for resources they have access to. These resources including source code, SQL queries, cron files, or script commands.
- View their own push requests
- Edit push requests they have created before they are pushed
- Browse the push requests of other users pushing resources that they have access to view including post-push results
- Validate or unvalidate push requests that have been submitted to resources they have access to (submitted by themselves or another user)
- Alter their push user profile
- Define whether to receive email messages from the system or view them online through the tool
- View received messages and send messages to other system users

3.1.1.4 Push Administrator

A push administrator is a user of the deployment tool that has full permissions to approve and process push requests, and edit or configure any system resources. The following are features available to Push Administrators in the system:

- Create/Edit push users
- Create/Edit resources to be managed (SVN locations, MySQL databases, Cron files, servers allowing script execution)
- View/send messages to other users and those sent by the system
- Manage the deployment and creation of Push Branches both for testing and production deployments (including the configuration of SVN externals)
- Create, edit, push, re-push, and validate push requests
- Create/Edit user groups
- Create/Edit SVN sources (projects)
- Create/Edit push database resources
- Create/Edit cron file resources
- Create/Edit destination groups
- Create/Edit push servers (including permissions to execute scripts on those servers)
- Create/Edit final destination (production) branches
- Configure the creation of automated test branches (** work in progress)

- Define push server types (allows for expanded functionality by creating custom server types)
- Configure global setting from the push tool including global email settings, number of revisions to show when creating push requests with SVN resources, and a list of email addresses to be notified when branch pushes fail or succeed (used to notify IT or other external parties when production level changes are made)

3.1.1.5 Non-Registered User

Users not yet registered can submit a request to be added as a push user from the tool's main page. This message will be sent to the designated Push Administrator to take action on.

3.2 Deficiencies Addressed

As discussed in chapter 2, when analyzing the previous Doba deployment system, its processes have several shortcomings. The deployment tool system developed in conjunction with this research addresses each of these shortcomings in order to resolve the problems they present.

First of all, the deployment tool intrinsically creates a record of all resources that are deployed. This allows the records to be searched and analyzed in a much simpler and usable fashion than when they exist within a standard email archive.

SVN resources in the deployment tool are also pushed by revision number so that only the changes occurring within the specified revision are deployed, thereby eliminating the unintended deployment of resources that occurs when simply pushing the latest revision. Since all resources for a specific environment (development, staging, or production) exist in their own branch, they can easily be checked out to duplicate any of the environments for testing or system expansion.

Security concerns have been alleviated by the implementation of the "push/pull" method of having target servers update themselves, thereby eliminating the need for a central storage of credentials for all target systems.

Since all resources are managed by the deployment tool, when the developer submits them, the deployment tool will do some preliminary validation, and then thereafter deploy to each environment in the same fashion, eliminating any error that occurred in the past due to cut-and-paste operations or alterations made by an email client. For SQL changes, the same SQL is run in both staging and production, and for SVN resources, the entire contents of the staging branch are merged into production, ensuring that the exact same resources that were deployed to staging are also deployed to production.

The introduction of business logic also occurs within the deployment tool. While consisting of fairly simple business logic, the tool allows for QA (Quality Assurance) validation to be done on push requests before they are deployed to production to help ensure they have been adequately tested.

Cron file changes are also managed by the deployment tool by including them in the resources versioned by SVN. This allows all changes made to automated processes to be tracked and validated just as any other resource change.

In direct response to the amount of time required of the deployment manager in the old Doba deployment process, the deployment tool provides a simple interface for selecting and deploying resources with only a few clicks instead of several manual steps. This conserves the time of a skilled worker while also greatly reducing the chance for user error.

3.3 Design Methodologies

This document contains design descriptions for code implementations to create all necessary web pages, objects, and database elements for the automated code push tool. Reuse of various elements has been considered to help increase efficiency and quality. The system has also been designed to be scalable to accommodate the additions of new servers and resources, and the alteration or expansion of the push process.

The design consists of diagrams and descriptions of the UI functionality for all pages, and a diagram of the database design that will be utilized by the tool. In addition, there is also a list and description of all required Objects that must be created to interact with the various servers and web pages.

3.3.1 Persisters

An important consideration in the design process is the use of "Persister" objects that serve mostly to separate database functionality and SQL code from the business logic code. Therefore, all objects in the system that allow persistence have a corresponding persister object. The naming convention for object persisters is to name the persister with the object name followed by "Persister". For example, for the "PushRequest" object, its persister is named "PushRequestPersister."

3.4 Utilized Technologies

This section describes the various technologies utilized by the code deployment tool in this research and how they are used. These include Subversion, PHP, Smarty, MySQL, AJAX, and DHTML/JavaScript.

3.4.1 Subversion (SVN)

The keystone technology, and most important component of the research is the Subversion versioning system, also know as SVN. Critical to the functionality of the system is the leveraging of the branching feature in SVN. Used as a means to segregate the production version of resources from the various possible testing versions, the efficient branching and merging functions in Subversion provide the mechanism for effectively managing requested versions and deploying them to the various environments.

3.4.2 PHP

In order to drive the code deployment tool's web interface and back end scripts that connect it to the other components, PHP has been utilized. It was selected because of its robust functionality, rapid development, and ability to customize interactions with the host server OS, which is has proved essential for running scripts and interacting with Subversion. Another important factor in selecting PHP as the scripting language of choice is that it is open source and interacts wells with other open source applications that provide the core for a large percentage of web application systems (PHP, 2007).

3.4.3 Smarty

Equally as important as the separation of business logic from persistence code, is the additional need to separate display code from that business logic. This is accomplished by utilizing the Smarty templating engine for PHP, which consists of a set of PHP Objects that are used to create templates. These templates are then used to dynamically generate PHP pages on the server before serving them to the requesting client. The use of this system allows an implementation where the core PHP objects and business logic pages can be built without having to embed any HTML. Instead, data that is generated by these scripts can be assigned to variables that are then passed to the Smarty templates which then decide how to display them (Smarty, 2007).

3.4.4 MySQL

Used to persist settings, results, and other data pertinent to the code deployment tool, MySQL is an open source relational database application that is optimal for the research. In addition to its core functionality in the deployment tool itself, MySQL is also currently the only database system that can be utilized by a resource being managed by the deployment tool. This provides an adequate demonstration of the functionality of this research, since it is one of the most commonly used database systems in web based applications of small to medium size (MySQL AB, 2007).

3.4.5 AJAX

In order to provide more dynamic, application-like functionality from the deployment tool, and prevent inconvenient reloading of pages when interacting with several back end components, the deployment tool utilizes AJAX (*Asynchronous Javascript and XML*) functionality, which consists of the use of client side JavaScript calls to server side resources to retrieve data or enact processes without having to reload the entire web page.

Facilitating the implementation of this functionality is the use of an open source third party AJAX library called *YUI (Yahoo! Interface Library)*, which provides the framework for making AJAX calls (Yahoo! UI Library, 2007). The selected library is maintained by Yahoo!, and can be downloaded free of charge for integration into custom web systems. It provides a framework that handles all the client side JavaScript for creating an asynchronous connection to the server, listening for and handling the response, and processing it according to success or failure.

3.4.6 Dynamic HTML/JavaScript

Used in concert with AJAX functionality to provide a more application like feel to the deployment tool, Dynamic HTML (DHTML) and JavaScript are heavily utilized. These are generally used on pages that require dynamic user input and are used to collect that input before submitting it to the server for processing.

3.5 Use Cases

There are four main actors that participate in the deployment tool system. These are the Push User, Push Administrator, Destination Cron, and Deployment Tool, with the latter two being system actors. This section outlines the roles and functionalities of these various actors.

3.5.1 Push User

The most active user by volume, the Push User is the everyday user of the deployment tool system who creates changes in the various managed resources and submits push requests to be processed to move his changes to testing and production environments. This user must first commit their resource changes to the SVN project trunk, and then create a push request to push those resources to the desired testing environment. The Push User also has access to configure his own user settings for the deployment tool. Figure 3-1 below outlines the use case for this user.

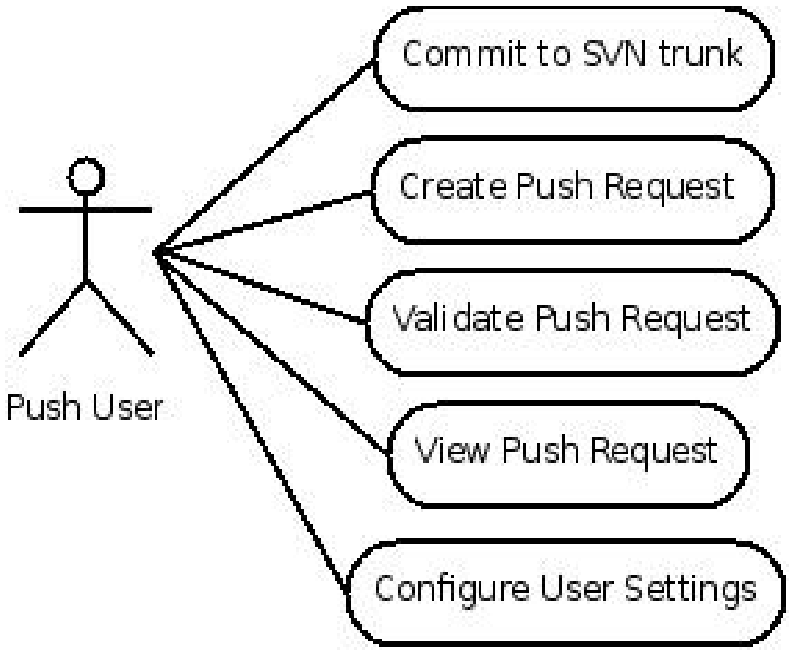

Figure 3-2 Push User Use Case

3.5.2 Push Administrator

The Push Administrator is a role held by the push manager or user of the deployment tool that will process push requests and administer requests for the other users of the tool. Such administrators have the most access and therefore the most control over the functionality of the system. The first responsibility of this user is to configure all of the resources that are to be accessed and managed by the deployment tool system. This includes the addition and configuration of Push Users, user groups, databases, push servers, cron scripts, and destination as well as testing push branches. Once push requests have been submitted by Push Users, the Push Administrator is responsible for processing and approving push requests, and once validated, subsequently processing the pushing of test branches to their destination

production branches. Figure 3-2 below shows the use case for the push administrator user.

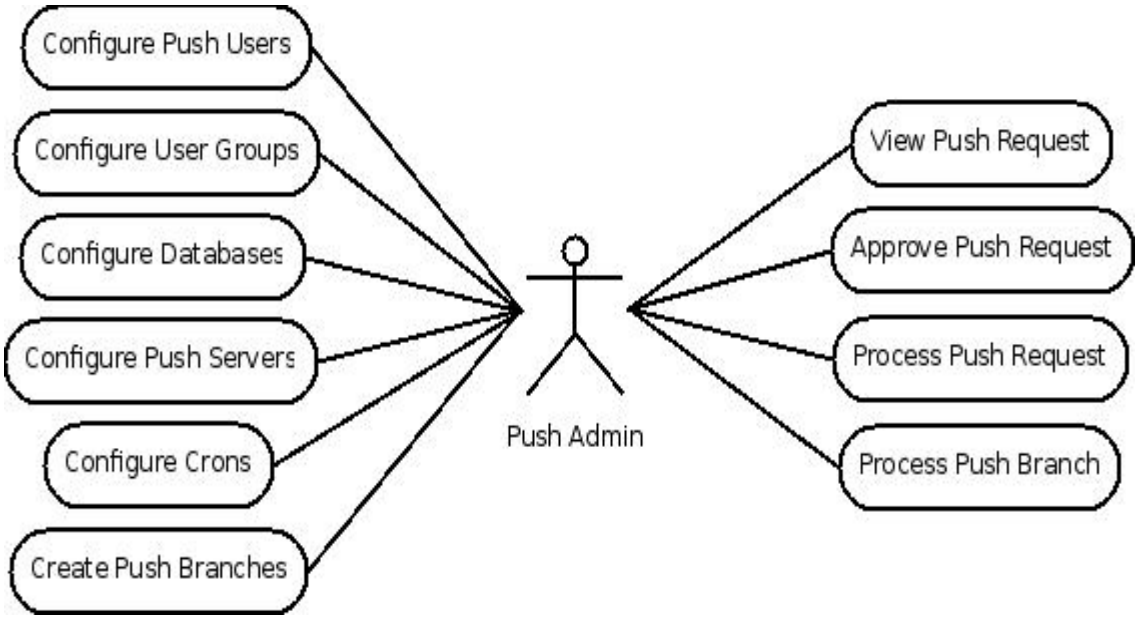

Figure 3-3 Push Administrator Use Case

3.5.3 Destination Cron Script (System)

This actor is a system role that is run remotely from the main deployment tool system on each of the destination servers where code is pushed . Its responsibility is to manage the updating of push resources on its respective host according to the settings dictated by the main deployment tool. Upon completion of these tasks, which can include updating a working copy or running a script, this actor will store the results on success or email the admin on failure. The use case for this actor is outlined in Figure 3-4.

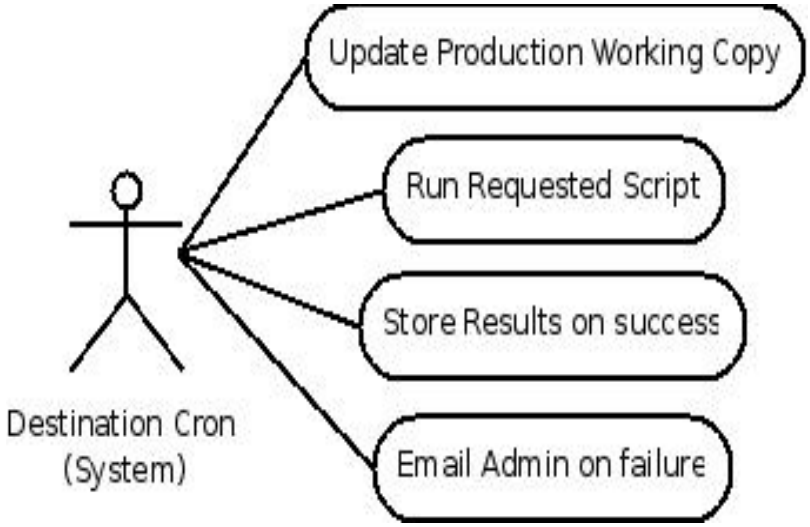

Figure 3-4 Destination Cron (System) Use Case

3.5.4 Deployment Tool (System)

The most active system actor is the Deployment Tool actor. This role has the responsibility for handling the backend tasks of the deployment process including the merging of push request changes into the specified testing branch, the processing of SQL push request resources, changes to the cron resources, the queuing of scripts to be run on target push servers, the creating and processing of push branches, and the notification of system users when events happen within the system. Figure 3-5 below outlines the use case for this user.

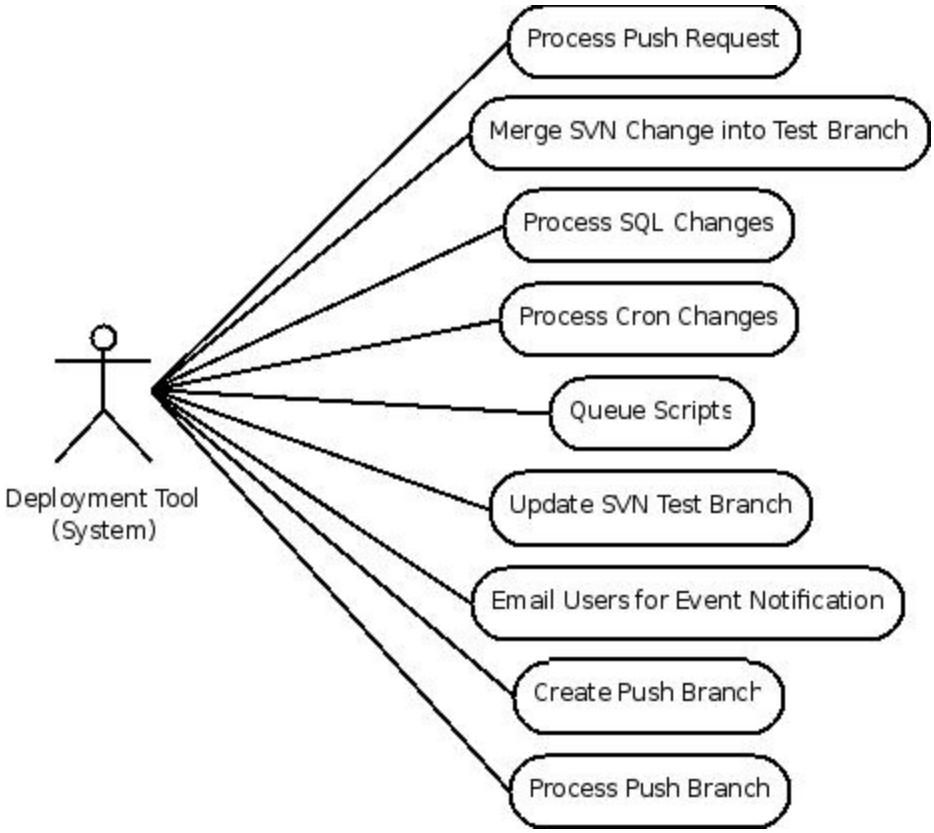

Figure 3-5 Deployment Tool (System) Use Case

3.6 Constraints

With the current implementation, the system is constrained to use Subversion (SVN) as the versioning mechanism for managing source code resources, although the architecture was designed to be intentionally flexible to allow for the future support of alternative versioning systems. The push user must also manage initial resource commits to the versioning system through an external means (a separate Subversion client), as this is not manageable through the deployment tool. To some extent the push user must also keep track of the revision numbers that are committed, as they must select them from a list of recent revisions in order to specify which files to push when creating a push request.

Database resources are currently limited to those utilizing MySQL, although similar to the architecture of the versioning portion of the system, database functionally has been architected such that it is easily expandable. Database versioning is also not currently handled by the tool, and therefore if desired, must be managed by a separate external mechanism such as the database platform itself.

During the system setup process, there are some manual tasks that may need to be performed for the system to function correctly. The destination cron scripts must be installed and configured on each destination server. This includes granting the necessary access to the deployment database from each of these servers so that these crons can report their performance to the system. Also, if the SVN repository is utilizing SSL encryption with HTTP authentication and its certificate is not signed by a valid Certificate Authority (CA), the server running the deployment tool will need to be configured to accept this certificate.

Due to time constraints on the number of resource hours to be allocated to development, access to the deployment tool is also currently only supported via Firefox, which is available on all of the most common platforms. Making the tool cross-browser compatible is not necessary for the research to be successfully tested and evaluated.

3.7 Business Logic For Release Management

Integrated into the code deployment tool is the ability to use release management business logic to handle quality assurance tasks as part of the deployment process. This currently involves the ability of the *Push*

Administrator to require all push requests added by other *Push Users* to be flagged for admin approval before they can be processed. This can be done on a per-user or per-destination branch basis. For security purposes, by default all push requests containing scripts are also flagged in this fashion.

In addition to the role played by administrators, other users also participate in the process through a simple approval process before push requests are pushed live. This involves the user validating that his changes are correct in the testing environment (which contains resources from the testing branch that his push request was pushed to), and then marking his push request as validated. If the *Push Administrator* tries to merge the testing branch into its final destination branch before all push requests have been validated, he is warned that not all push requests are validated and he can either ignore the warning and push anyway, or take action to ensure that the changes have been properly validated.

3.8 User Interface (UI) Design

As is common in solid user interface design, the interface for the code deployment tool has been designed to be as concise and simple as possible. In addition, UI functionality has been reused wherever possible to prevent duplicate functionality. While no formal usability testing was conducted, user feedback was solicited and interface improvements were made as a result. It is possible that further usability testing would result in increased system efficiency and user satisfaction.

3.8.1 UI Flowchart

There is one main division in the user interface of the deployment tool that is defined by the type of user is currently logged in. The two types of possible users are normal push users and administrators. Push users have access to a subset of the components that are accessible to push administrators, as defined in Figure 3-6 below.

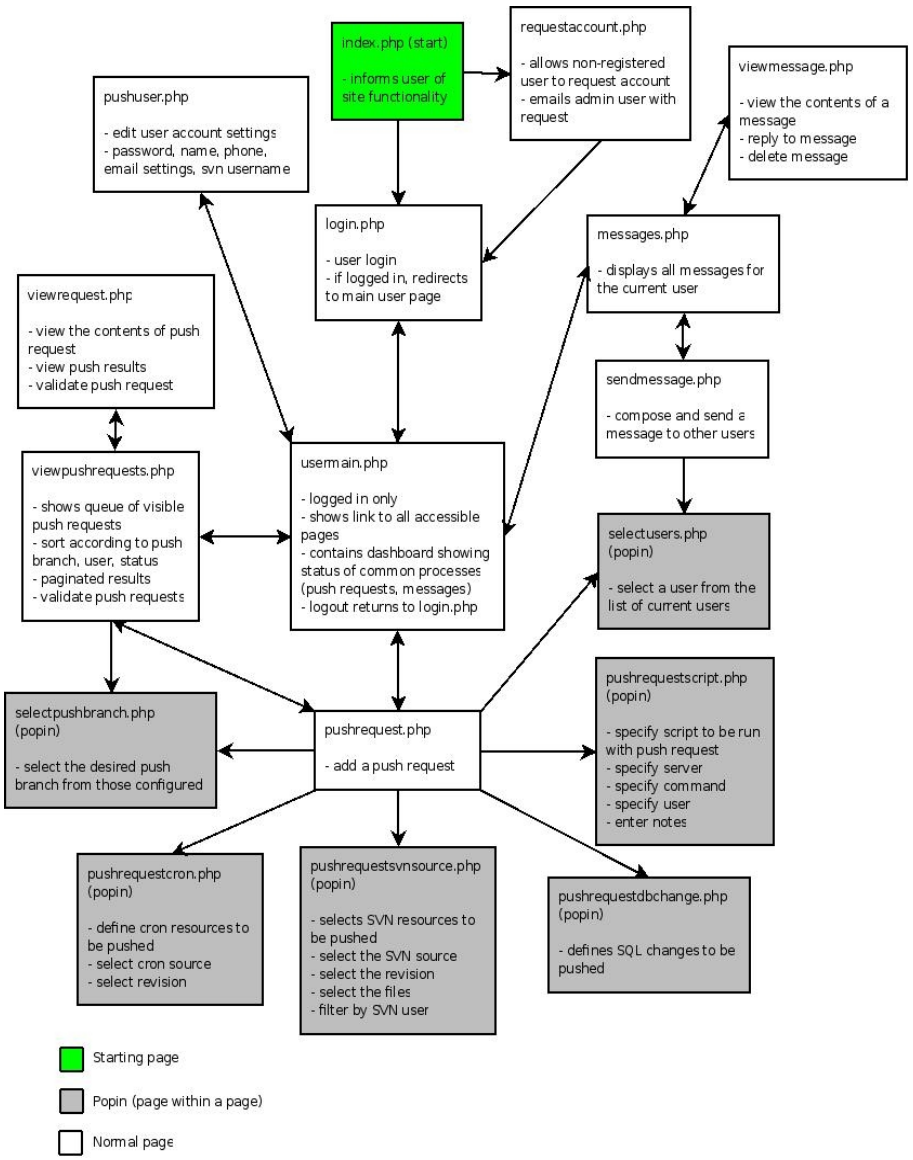

Figure 3-6 Push User UI Flowchart

Push Administrators have access to all UI components in the deployment tool, including all those available to normal push users since they are also able to create push requests to be submitted to the system. The following, **Figure 3-7**, is a flowchart describing push admin access in the deployment tool system.

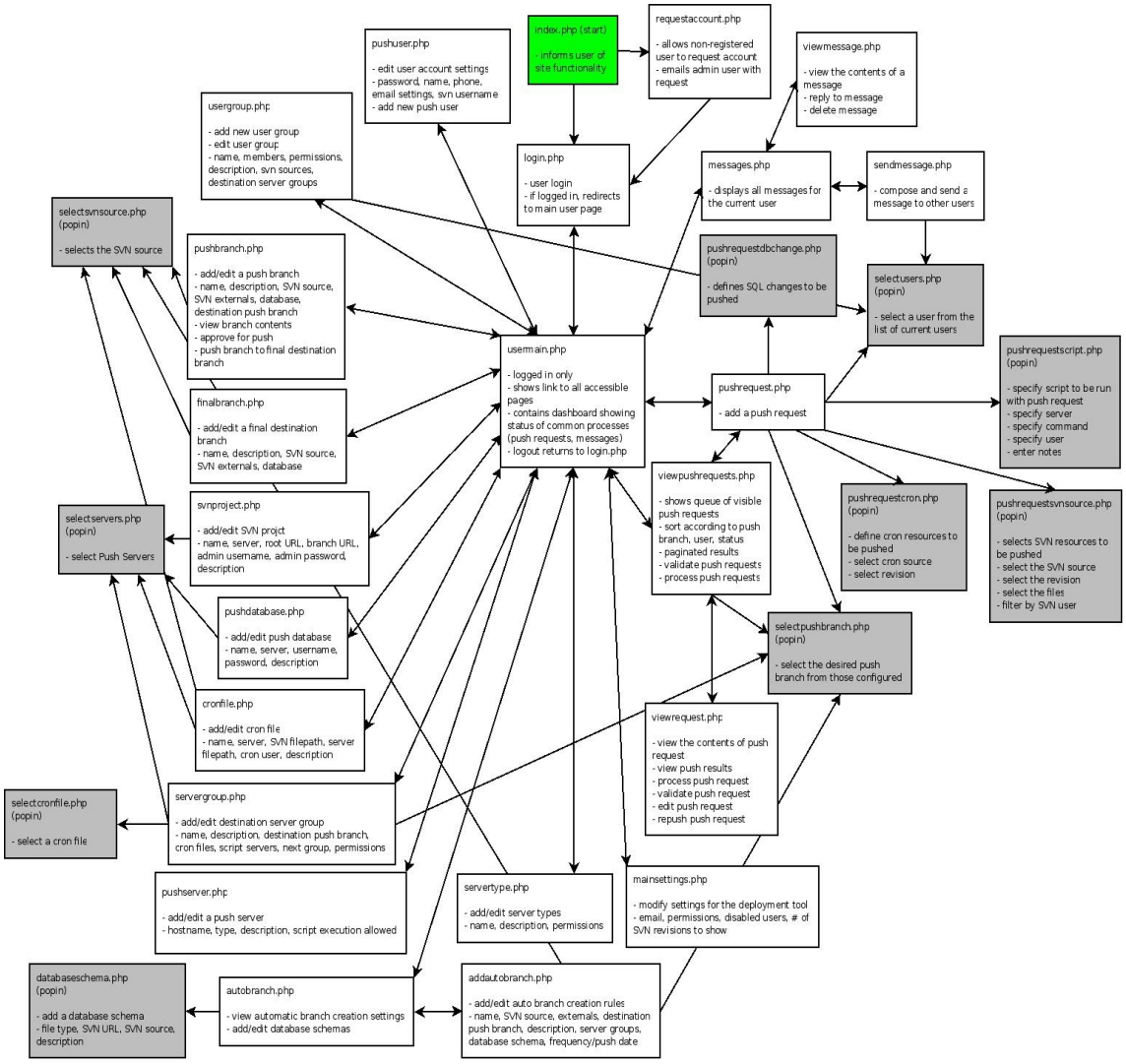

Figure 3-7 Push Administrator UI Flowchart

3.8.2 Screen Shots

This section contains screen shots of key elements of the deployment tool system and a brief description of each. A “popin” refers to an iframe that appears within the browser window (as opposed to a popup which appears outside the window in a separate browser instance) and facilitates some functionality UI functionality that exists within the context of the parent page.

3.8.2.1 User Main Page

This page is what a normal push user sees when they log into the deployment tool. It acts like a dashboard for system functionalities that they have access to.

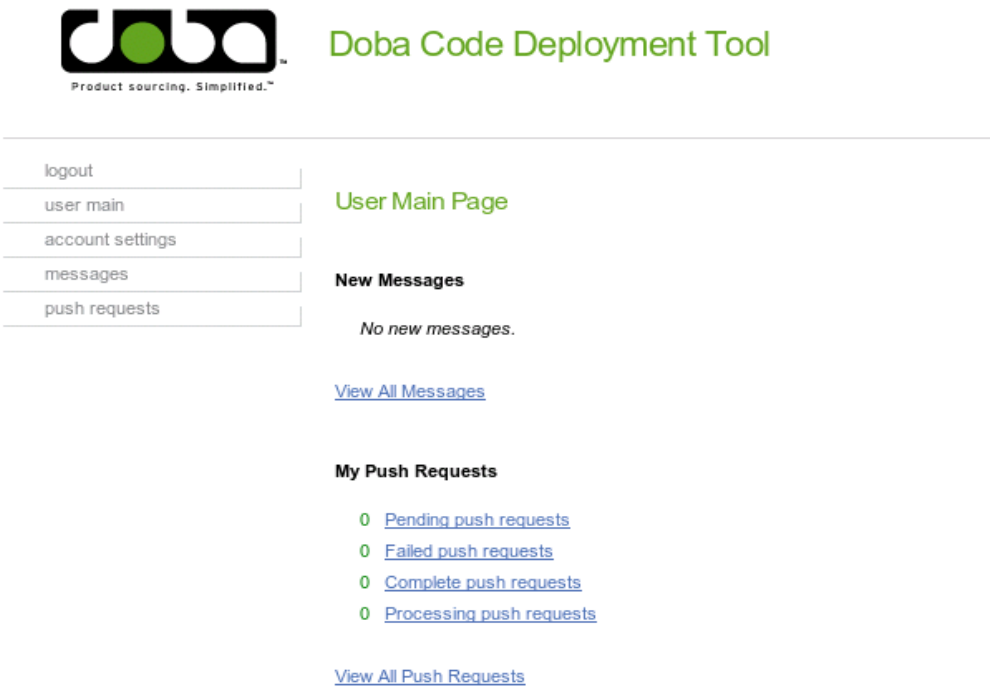

Figure 3-8 User Main Page

3.8.2.2 Add Push Request

The following screen shot in Figure 3-9 portrays the page used to add push requests into the system. It includes necessary data for defining the push request and its contents. Using the links on the page, the user can dynamically define any number of push resources to be included in the push request. These resources can be SVN files, database changes, cron file changes, or scripts to be run.

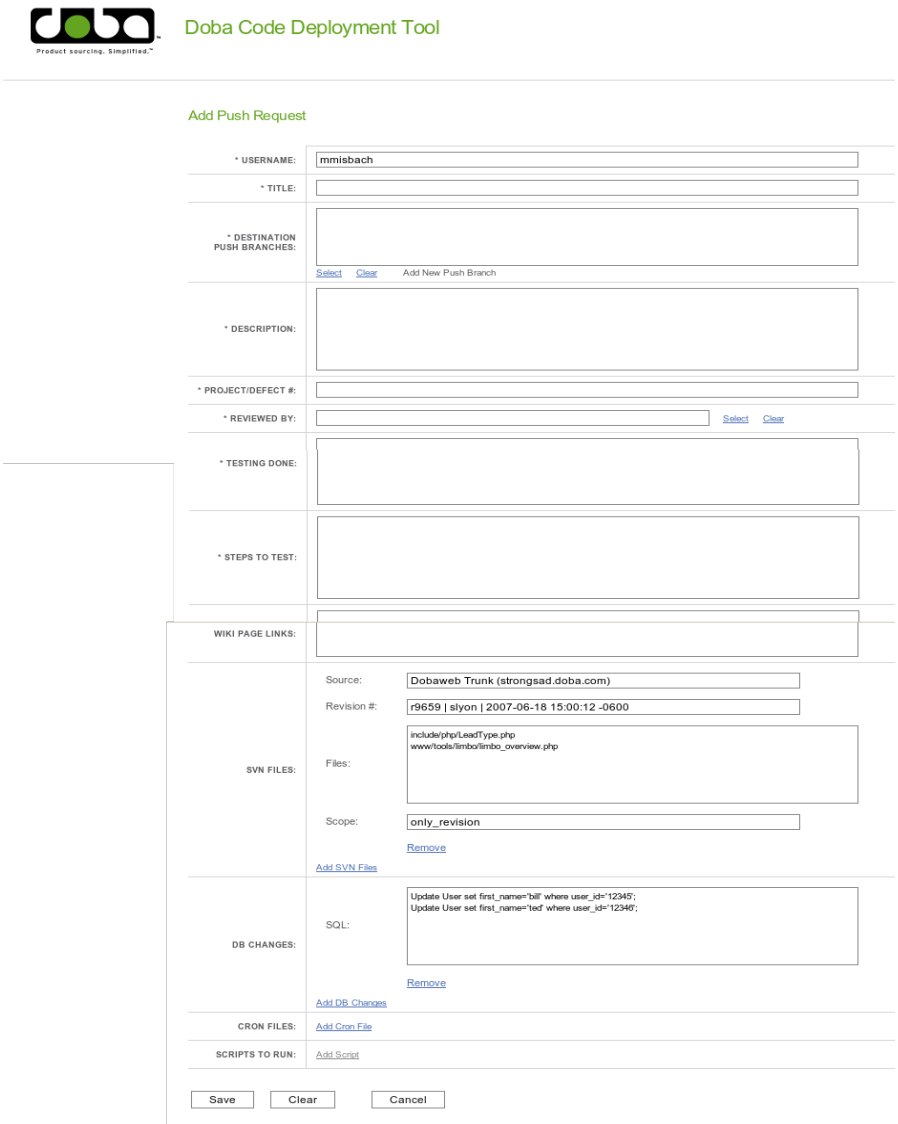

Figure 3-9 Add Push Request

3.8.2.3 Add SVN Files (popin)

This page is a popin (a dynamic frame similar to a pop-up, except that it exists within the page being viewed) that exists as part of the Add Push Request page and is used to specify SVN files for addition to the push request being created. The user can select the SVN source repository, and the revision number, and details about that revision will be displayed.

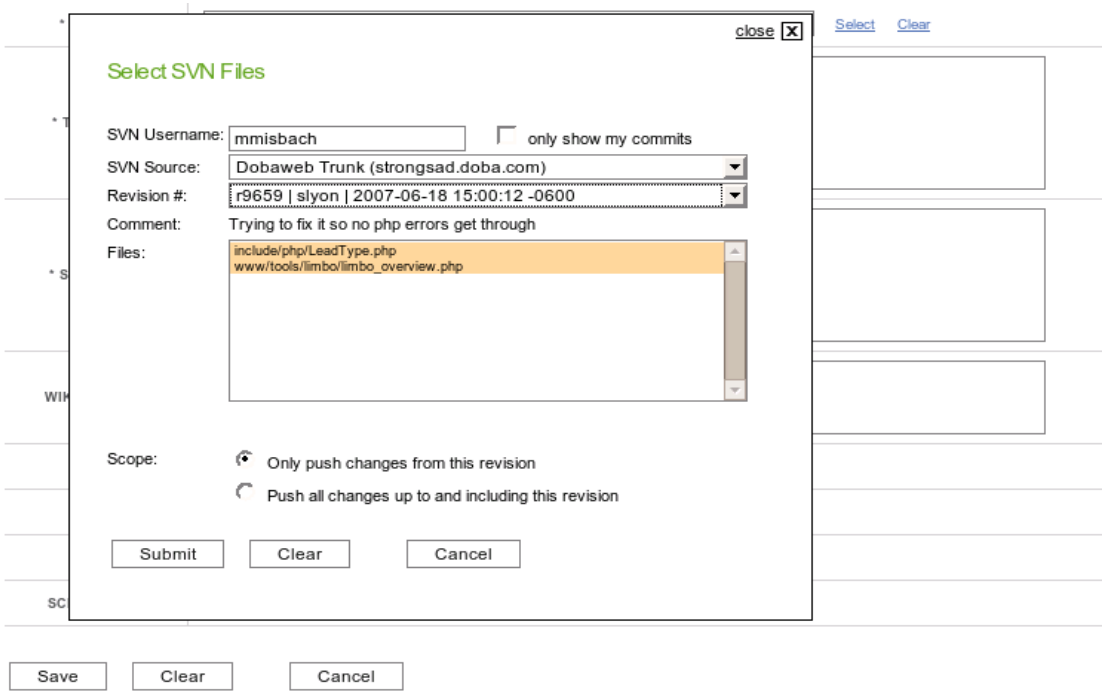

Figure 3-10 Add SVN Files (popin)

3.8.2.4 Add Database Changes (popin)

Likewise, this screen shot shows the popin used to add SQL changes to a push request. It is a simple page that allows the user to specify the SQL statement or statements to be executed.

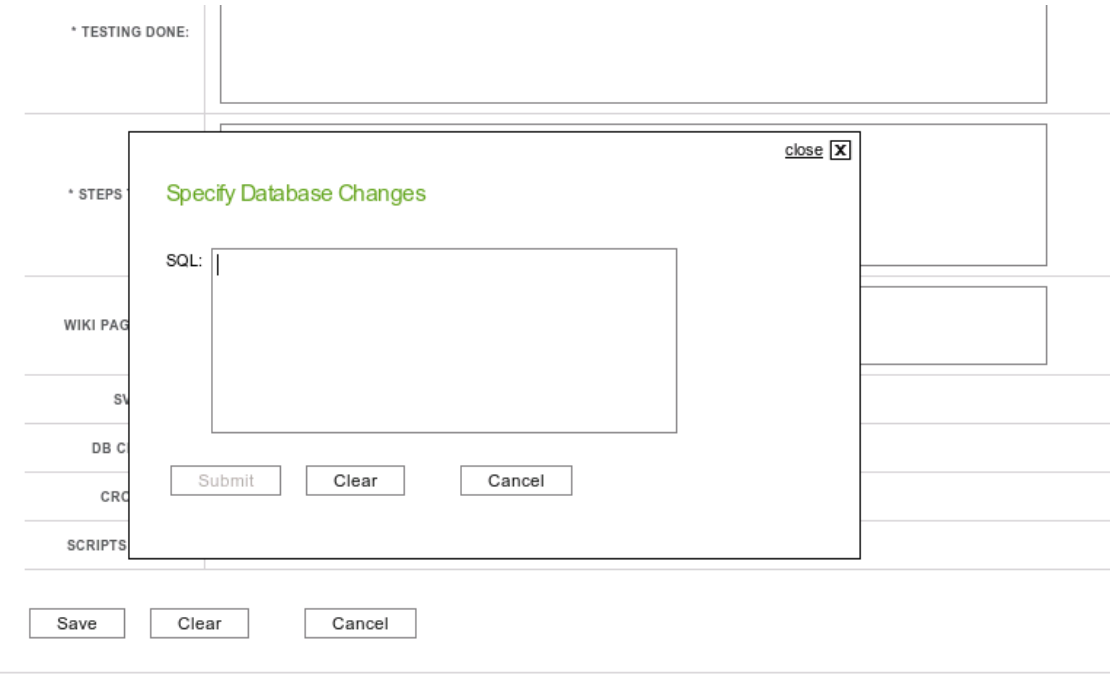

Figure 3-11 Add Database Changes (popin)

3.8.2.5 Add Script (popin)

This page is also a popin in the add push request page and is used to add script resources to the push request being created. The push user specifies the command to be run and then selects the server where it should be executed. They must also define the user it is to be run as on the target server, and notes describing the purpose of the command. These notes are necessary since by default all push requests with scripts must be approved by the push manager before they will be processed.

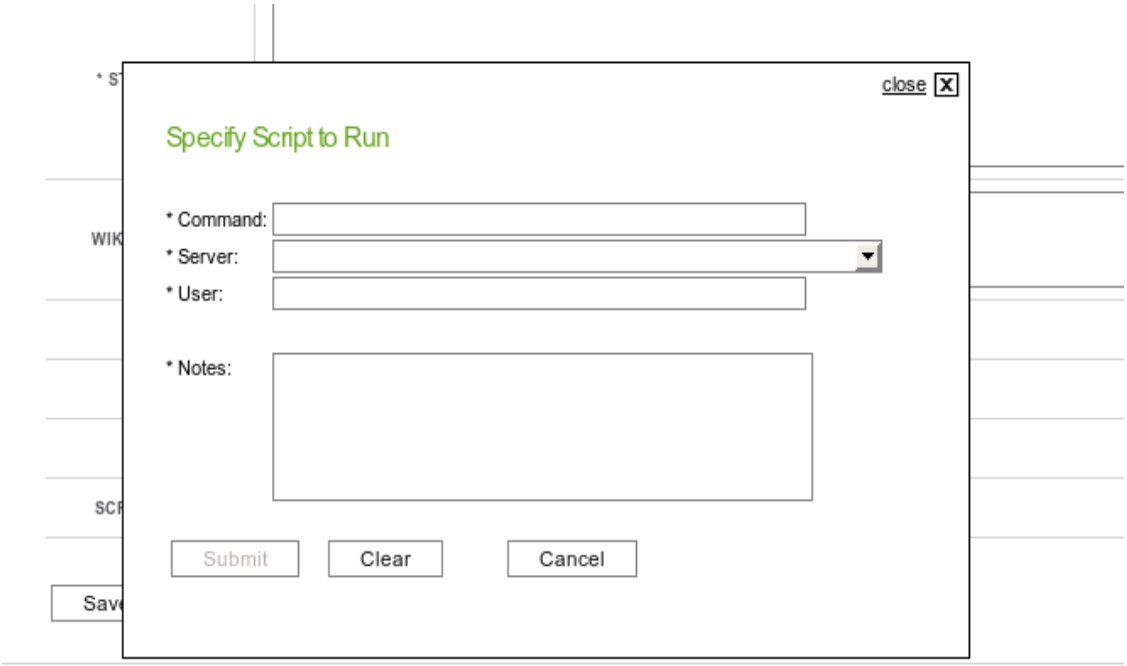

Figure 3-12 Add Script (popin)

3.8.2.6 Administrator Main Page

This is the page viewed by administrative users upon login and serves as their dashboard of functionalities within the deployment tool. In addition to having all the same functionalities as a normal push user, the push administrator also has access to all of the logging and configuration settings within the deployment tool.

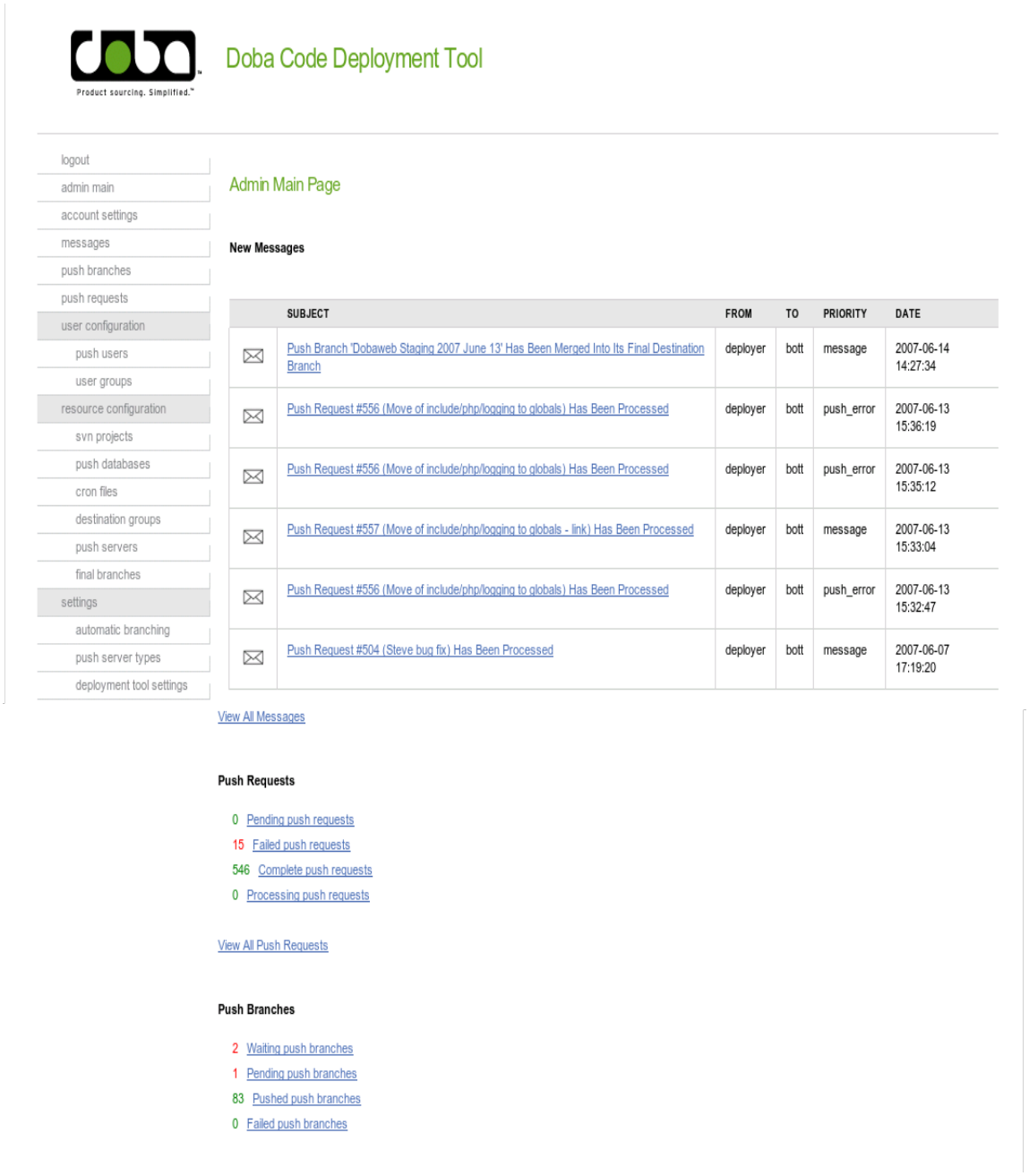

Figure 3-13 Administrator Main Page

3.8.2.7 View Push Requests (Administrator)

The following is the administrator version of the page used to view push requests in the system. Normal push users have access to the same page, but it provides less functionality for interacting with push requests.

They are not allowed to push them or edit them unless they are the author. Normal users can however, validate push requests requiring validation.

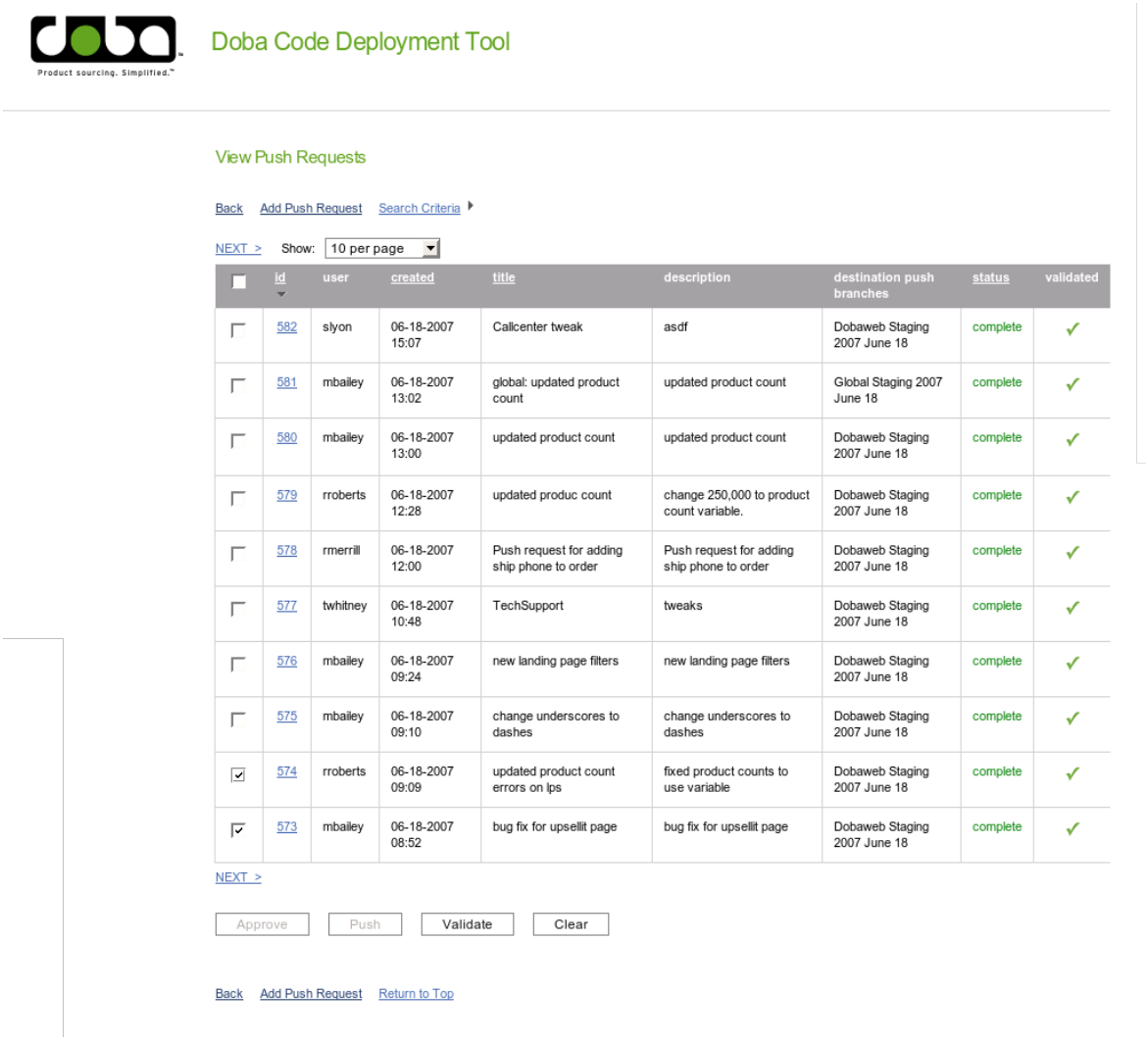

Figure 3-14 View Push Requests (Administrator)

3.8.2.8 View Push Branches

This page is visible to administrator users and gives an overview of the push branches existing in the system. It describes the status and source of each branch that is being managed by the deployment tool.

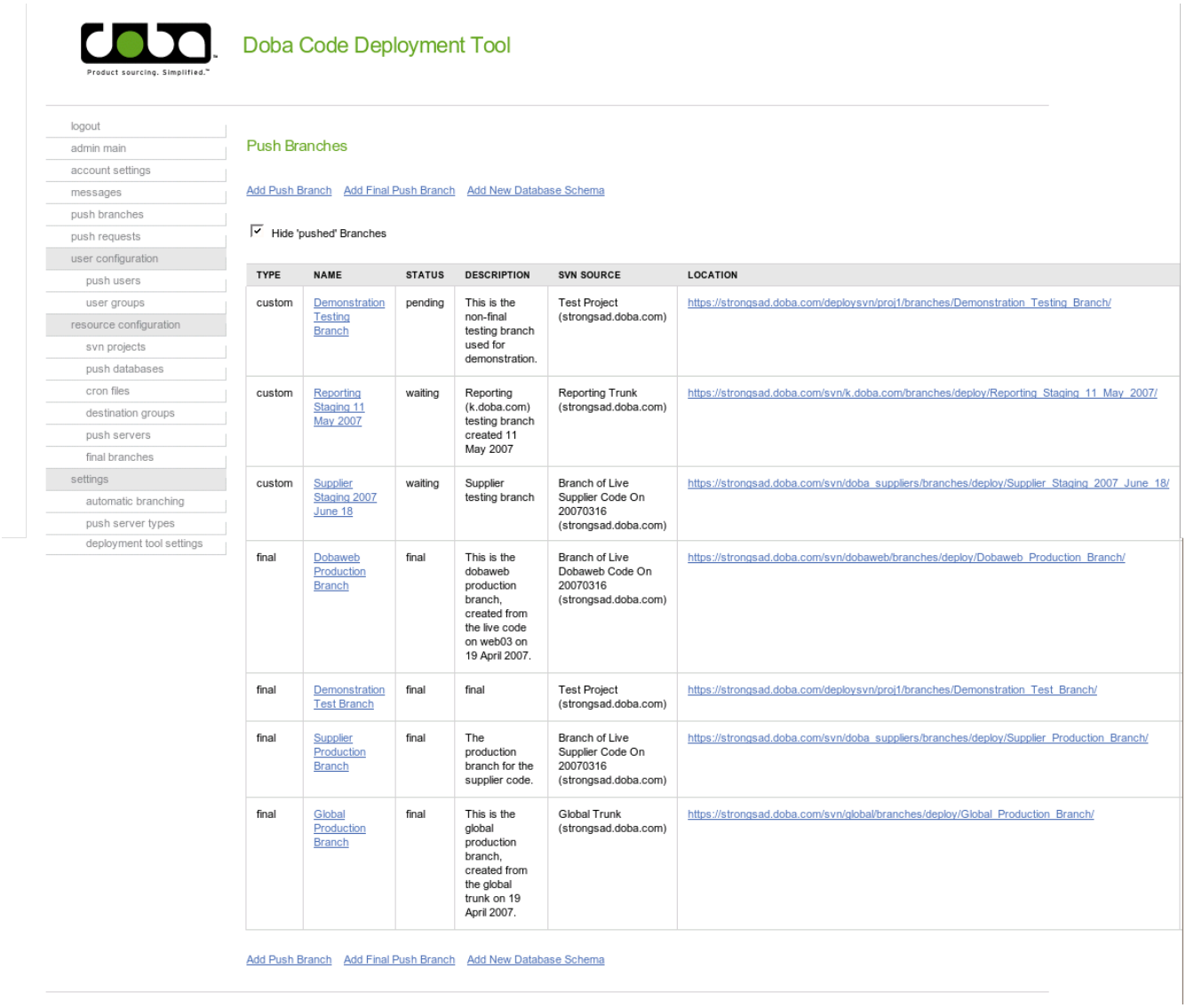

Figure 3-15 View Push Branches

3.8.2.9 Push Branch

This page is only visible to administrators and provides details on the configuration of a push branch being managed by the deployment tool as well as providing elements to be able to interact with it. These interactions include approving it for push, pushing it to its final destination branch, and modifying its settings.

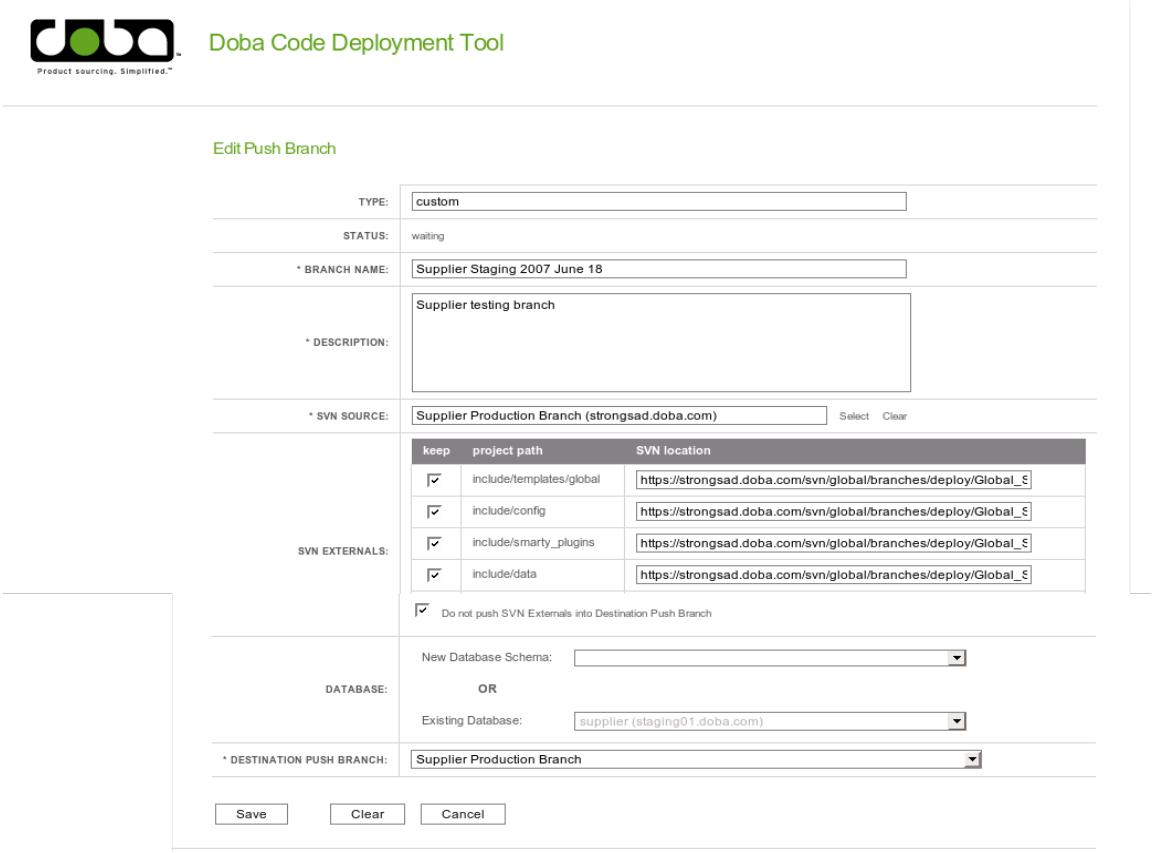

Figure 3-16 Push Branch

3.9 Database Design

In order to store the push request and configuration data needed by the deployment tool, a relational database is used. This relational database utilizes MySQL and has been normalized to include 31 separate tables. Each table corresponds to a business object with the same attributes and a matching persister which contains the SQL statements needed to store and retrieve records from the table.

The database schema has been normalized to allow for flexibility in managed resource configuration, as well as optimization for speed and space usage. In addition to the normalized schema, the database has also

been created with indexes designed to optimized query times for each of the tables, and have based upon the ways that records in each table are accessed.

The full schema diagram, a more detailed description of each table, and the actual SQL statements used to create and populate the database can be found in Appendix A: Database Schema.

3.10Core Objects

This section outlines the core PHP objects used in the deployment tool and includes a brief description of their functionality. Some of these objects correspond to database elements, while others involve business logic or system functionality needed to interact with outside systems such as SVN or MySQL.

For objects corresponding to tables in the database schema, there is an accompanying persister object which contains SQL statements used to persist objects to and generate them from records stored in the database. In addition, these business logic objects may also contain attributes that are arrays of sub objects. For example, the *PushUser* object has an attribute to store the *PushUserGroups* that the user belongs to. These can be modified and when the parent object is persisted, these sub objects will in turn be persisted as well through a call to their designated persister object.

Of the business objects, perhaps the most noteworthy are the *RepositoryConnect* and *SvnConnect* objects. Neither of these objects corresponds to a table in the database, but they are closely correlated and are vital to the functionality of the deployment tool. The purpose of

RepositoryConnect is to provide some compartmentalization for the future integration of different versioning systems. It is mean to act as an interface, and contains a reference to a separate connection object, whose type is defined when the object is used. In the current state of the tool, the only option is a SVN connection object *(SvnConnect),* since only Subversion will be supported initially. Important functions are defined in both and *RepositoryConnect* is the object used to interact with the versioning piece of the tool. When a function is called in *RepositoryConnect*, it simply calls the same function in *SvnConnect* and passe Appendix A: Database Schema s it the same parameters. This allows for future objects to be added to handle other types of versioning systems.

A full listing of all objects in the system as well as a description of each, can be found in Appendix B: System Objects.

3.11 Resource Configuration

In addition to the functionality of the deployment tool on the server side, an integral piece of the deployment tool is the ability of target servers to receive the changes and commands demanded by the system. For security reasons, the system was designed so that these target (usually production) systems would pull down the changes, thereby segregating them from most security issues that may arise within the deployment tool itself. Thus when functionality requires changes to be made on target systems (such as updating code or running a script) the approach has been geared towards the target pulling down those changes rather than having them pushed out to the server. This “push and pull” model provides a significantly more

secure means of deploying resources to sensitive servers, than by simply pushing them out from the central deployment server, as that would require the storage of login credentials to connect to them all.

In order to accomplish this, a series of scripts have been written for each of the necessary tasks. The target system hosts these scripts and they are scheduled to run at a regular interval and handle pulling down the requested changes as well as writing back the results to the main deployment database. These scripts are to be used for pulling down SVN resources, cron file changes, or running scripts on a target machine and are described below.

3.11.1Destination SVN Server

Each destination server that will be hosting SVN resources will run this code package. The *processsvnupdates.php* script should be scheduled on the destination to run every minute so that it can quickly process any updates that occur through the deployment tool to the working copies it manages. It will read configuration settings from *svnsettings.php* which will determine the location of working copies on the server and what user should be used to update them. These files can be found in their entirety in Appendix C: Destination SVN Server Scripts.

3.11.2Destination Cron Server

The purpose of this part of the tool is to process changes to managed cron resources, and is run by all destination cron servers. The code in this package will connect to the main deployment tool database, check for cron

updates, process them, and then store the results of the process. It consists of a configuration file called *cronsettings.php* and the main processing script called *processcronchanges.php* which is cronned on the host to run at a frequent interval (recommended every minute). These files can be found in their entirety in Appendix C: Destination Cron Server Scripts.

3.11.3Destination Script Server

Each server that the tool allows to run scripts will host this code package. It consists of *scriptsettings.php* and *processscripts.php*. The former contains the settings for the server as they appear in the deployment tool and the latter is the script that is cronned to be run at a regular interval. It functions by connecting to the main deployment tool database, checking for script submissions for the host server, processing them, and storing the results into the database. These files can be found in their entirety in Appendix D: Destination Script Server Scripts.

4 Results and Analysis

In this chapter, a description of the implementation of the design discussed in chapter three is presented. This includes a list of features, performance analysis, user experience feedback, and a comparison of system capabilities with a real world test case implementation done at a web-based software company called Doba. Doba is located in Orem, Utah and provides its customers with an online product sourcing web application that utilizes all of the types resources addressed in this thesis. It also provides an excellent venue for analyzing the results of applying the research in a real-world scenario. All performance analysis, user feedback and comparison data contained in this chapter come from observations taken from the application of the research within the Doba environment.

4.1 Doba Test Case

In order to conduct comparisons on the effectiveness of the research conducted, it has been practically applied to the web application system employed at Doba, a large online product sourcing company. The systems architecture at Doba matches the profile for a medium sized web application system as previously described, with multiple web servers serving static, dynamic, and relational database content. In addition, it has resources dedicated to performing automated processes to meet the business needs of the web application.

As discussed in chapter 2, prior to implementing the results of this research within the Doba systems, resource deployment was manually processed through remote SSH connections to the testing and production systems, introducing several weaknesses. Each of these drawbacks have been addressed and resolved in the research. For example the issue or unintended resource deployment as a result of always deploying the latest revision has been resolved in the research system by deploying SVN resources by revision number, and by default only pushing changes made in the specified revision.

The research system solves the issue of discrepancies in SQL changes between staging and production deployments by integrating the submission of SQL changes into the push request process so that they can be automatically processed by the deployment tool when the push request is processed, intrinsically creating a log of all queries run. This ensures that exactly the same query that is run in the testing environment is also run on production. Likewise SVN resources deployed to staging and production environments are exactly the same since the entire contents of the staging SVN branch are merged into the production branch when it is deployed.

A solution to resolving the issue of resource deployment logging has also been demonstrated in the research. The deployment tool implemented in the research creates an intrinsic log of all resources deployed that can be easily searched and analyzed, making it much simpler to troubleshoot deployment related issues.

4.2 Performance Analysis

This section describes the performance analysis that was conducted after observing the deployment tool functioning, deploying resources within the Doba testing and production environments.

4.2.1 User Response Evaluation

Most of the result analysis utilizes the formal feedback provided by system users in the form of a post-usage survey (see the ***User Feedback Survey*** section). There was also additional informal feedback provided by users, primarily during the development and testing phase, which was translated into alterations to the functionality of the deployment tool's user interface. The results of the formal survey can be seen later in this section.

4.2.2 Push Request Submission Process

Performance improvement is primarily concerned with the point of view of the Push Users (developers) who are creating and submitting push requests. Instead of having to generate an email with a list of resources to be pushed, these users can simply use the dynamic web form to create push requests. It allows them to select SVN resources by specifying the source and selecting the revision number and files from an AJAX driven interface. SQL changes can also be entered and cron files specified for push.

After resolving some initial bugs in the process, including a significant one relating to the performance and speed of the AJAX used to populate the recent revisions and affected files when adding SVN resources, according the results of the post-usage survey, the response from users has been

positive in that the push process is now more streamlined. Since they only have to submit a single push request for both testing and production (the same resources are intrinsically pushed for each), this saves the user a step from the previous method, where they also had to email a production push request. Informal feedback from users also included the benefit of being able to view the push requests submitted by other users to determine what other resources were being concurrently deployed. During the testing phase of the deployment tool there were some requests for additional features such as being able to enter the revision number for SVN resources instead of selecting it from a drop-down list, and the ability to copy and edit a push request to easily submit a similar one. With the addition of these features, overall developer response has included increased efficiency and functionality.

4.2.3 Push Request Processing

From the perspective of the push manager, the process is faster, more efficient, more reliable, more stable, and more accurate from what it was previously. No longer does the push manager have to manually manage resources to be pushed. There is no more cutting and pasting from emails, worrying about spelling errors or incorrect path names, no more removing of email client injected new lines, replacement of slashes going in the wrong direction, or manual SSH sessions, thereby reducing errors. The deployment tool provides an all inclusive interface for processing push requests, and the bulk of the responsibility for correct resource specification has now been placed on the push request submitter.

Another important implication of the use of the deployment tool is the prevention of partial pushes. This is where some files will get pushed and others not, or a database query is pushed followed by file pushes that fail. With the old deployment process, these resources are pushed independent of each other, so if one fails, the other may still be deployed, which is not atomic. With the implementation in the research, since pushes are managed by the deployment tool, files are pushed first and if they fail, other resources that are part of the same push request are ignored. Since conflicts in the merging of files are the most common errors encountered, this prevents most of the push-related resource discrepancies. Should the SVN files be deployed successfully and the SQL queries fail, there will be a discrepancy until the SQL statement has been corrected and re-run, but this will only happen in the testing environment, which does not require synchronized resource pushes. The SQL can be corrected, and since the exact same commands are executed during the production push, by the time it is made, the correct statement will exist in the system and the code and SQL will be deployed together without error.

The push process is not without issues. At times conflicts may occur when merging in requested files which will cause the push request to fail. Thanks to the atomic commit functionality of Subversion, this does not create a partial commit situation in the destination branch however. Since files in a push request are first merged into a temporary working copy and then submitted to the target branch, if the pushing of resources to the testing branch fails, the temporary working copy will be discarded and the

push request marked as failed without making any changes to the target testing branch.

The most common reason for a file conflict is the existence of an unmergible revision between the current version in the destination branch and the revision being pushed. Since the default is to only push changes from the requested revision, this can sometimes cause conflicts if a missing revision has changes affecting the same file. If changes from all previous versions of the file are desired in the push, then the simplest solution is to change the scope of the push to include all revisions up to and including the one being requested. If not all revisions should be pushed, then all of those desired must be pushed individually through the deployment tool.

Overall, according to the results of the user survey, instead of spending up to two hours pushing resources to a test or deployment server, the push manager is now able to do it in minutes since he can usually just push a button and let the deployment tool do most of the work.

4.2.4 Increased Push Quality

This section describes several of the key features of the research that have contributed to increased push quality and reliability. These items include the use of the revision number in the push request, the fact that the same resources are pushed to testing and production, and logging of pushed resources.

4.2.4.1 Pushing By Revision Number

One important benefit of the implementation of this research is the alleviation of the issue created by always pushing the latest revision from the trunk as was previously done. Instead, specifying the revision number to be pushed allows the exact changes required to be deployed, regardless of what other users may be doing in the repository. There are two separate scopes in which the revision may be pushed. The default, as discussed previously, is to only push the changes from within the specified revision. The alternative setting is to push everything up to and including the chosen revision number. The utilization of the revision number as a criteria for push requests has allowed the system to more fully utilize the functionality of Subversion and prevent code changes from conflicting with one another within the push process. The assurance that only the specified changes will be pushed has greatly increased system reliability.

According to the results of the user survey, prior to the implementation of the research within the Doba test case, it is estimated that on approximately every 10^{th} push, resources were deployed unintentionally due to the practice of pushing the latest version from the trunk. At times when multiple projects are modifying the same resources, the frequency of error is increased. Since the deployment of the research, according to the Doba push manager, this has not occurred a single time and the number of push requests processed through the deployment tool at Doba is now over eight hundred.

4.2.4.2 Same Resources Pushed to Production

Because the deployment tool allows the storage of all push request information, it is simple to reuse that information when pushing resources to their final destination branch (production). Thus deployments to testing can be duplicated when pushing them to production. This is particularly applicable to SQL changes. These are replicated exactly when pushing to a production server because the tool stores the results of all requests that were processed in the testing environment.

Likewise the deployment tool ensures that exactly the same SVN resources are pushed on production as they were on testing, although it utilizes a different method to do so. This functionality is accomplished through the use of branching in Subversion. Since each testing environment has its own SVN branch, when it comes time to deploy resources to the production environment, the contents of the entire testing branch are simply merged into the destination one. Since the testing branch began as a copy of the destination one before any push requests were pushed to it, the only differences between the two are the push requests that were processed. This is much more reliable than the previous method of pushing SVN resources, thereby contributing to the reliability of the production resources. For a visual representation of how testing branches are merged into destination branches see Figure 3-1 Deployment Tool System Overview.

According to the results of the user survey, prior to the implementation of this research within the Doba test case, errors related to not pushing the exact same resources to production as were pushed to staging occurred approximately every fourth or fifth time pushes were made

to production. Most often this was the result of typos existing in the production push requests, or some resources previously pushed to testing that were not included in the production branch. Since the integration of the deployment tool into the push process, the survey of the Doba push manager reveals that this problem has not arisen.

4.2.4.3 Logging of Deployed Resources

The final increase in system reliability provided by the deployment tool is its intrinsic logging capabilities. It not only stores every push request submitted to it, but also logs the results of pushes both to the designated testing branch and also of the testing branches into their final destination branches. This allows auditing of changes made to testing and production systems, facilitating better debugging and roll back in the event of an error.

The existence of this log has proved beneficial since the inception of the research. Previous to its implementation, the full record of resources pushed was kept only by the push manager in the form of archived emails. This was not readily available to other developers, so when questions arose over what resources had been pushed and when, it was fairly cumbersome to track that down. Currently with the implementation of the deployment tool, all authorized users are able to view historic data on what was pushed. On several occasions this has saved the push manager significant time and resources, since developers were better able to track down their own issues. It was not measured exactly how much time has been saved, but there have been at least a half dozen times since implementing the research, that the

existence of a comprehensive log has empowered users to track down their own issues without any action by the push manager.

4.2.5 User Feedback Survey

In order to quantitatively gauge the amount of improvement that the research has provided to the Doba deployment process, a survey was conducted with users of the deployment tool, seeking to compare its functionality with that of the previous Doba push process. This provides some concrete data with which to analyze the improvements provided by the research.

4.2.5.1 Survey Contents

The post-usage survey developed to quantify the effects of the research is targeted towards users of the code deployment tool, which consists of two types, push request submitters and push managers. Since these different types of users perform different tasks, it was deemed important to distinguish between them. A push request submitter will have a better idea of what is involved in creating, submitting, and validating a push request, while push managers will have a better idea of the processing of a push request and the overall effect of the deployment tool on the push process. The survey administered consisted of the following 16 questions (Table 4-1 Survey Questions) as outlined below:

Table 4-1 Survey Questions

#	Question
1	Prior to the use of the code deployment tool, were you a push manager or just a push request submitter?
2	Are you currently a push manager or just a push request submitter?
3	Prior to the user of the code deployment tool, on average, approximately how many times per month were there SVN conflicts that ended up affecting production systems?
4	Since the implementation of the code deployment tool, on average, approximately how many times per month have there been SVN conflicts that ended up affecting production systems?
5	Prior to the use of the code deployment tool, on average, approximately how many times per month was code unintentionally pushed to production systems?
6	Since the implementation of the code deployment tool, on average, approximately how many times per month has code unintentionally been pushed to production systems?
7	Prior to the use of the code deployment tool, on average, approximately how many times per month were there SQL related issues on production systems?
8	Since the implementation of the code deployment tool, on average, approximately how many times per month have there been SQL related issues on production systems?
9	Prior to the use of the code deployment tool, on average, on a normal push day, how many minutes did you spend submitting, validating, and processing push requests?
10	Since the implementation of the code deployment tool, on average, on a normal push day, how many minutes do you spend submitting, validating, and processing push requests?
11	Do you think the implementation of the code deployment tool has made the Doba deployment process more RELIABLE? Rate this on a scale from ONE to SEVEN where ONE is much more reliable, FOUR is just as reliable and SEVEN is less reliable. Please explain your answer.
12	Do you think the implementation of the code deployment tool has made the Doba deployment process more STABLE? Rate this on a scale from ONE to SEVEN where ONE is much more stable, FOUR is just as stable and SEVEN is less stable. Please explain your answer.
13	Do you think the implementation of the code deployment tool has made the Doba deployment process more ACCURATE? Rate this on a scale from ONE to SEVEN where ONE is much more accurate, FOUR is just as accurate and SEVEN is less accurate. Please explain your answer.
14	Since the implementation of the code deployment tool, do you think the testing (staging) environment is more or less effective? Rate this on a scale from ONE to SEVEN where ONE is much more effective, FOUR is just as effective and SEVEN is less effective. Please explain your answer.
15	Are there any improvements you have seen made through the use of the code deployment tool not previously mentioned in this survey? If so, please list them.
16	Are there any drawbacks you have seen introduced by the code deployment tool that have not been previously mentioned in this survey? If so, please list them.

4.2.5.2 Survey Results – Raw Data

The post-usage survey was administered to a total of 8 users of the deployment tool, all of which are developers that use the Doba deployment process extensively. For answers where the user gave a range for their response, the largest value in the range was consistently used. The tables below, show the results for each user who took the survey.

1. Prior to the use of the code deployment tool, were you a push manager or just a push request submitter?

Table 4-2 Survey Question #1 Data

User Type	#
Push Managers	2
Push Submitters	6

2. Are you currently a push manager or just a push request submitter?

Table 4-3 Survey Question #2 Data

User Type	#
Push Managers	1
Push Submitters	7

3. Prior to the use of the code deployment tool, on average, approximately how many times per month were there SVN conflicts that ended up affecting production systems?

Table 4-4 Survey Question #3 Data

Respondent	Answer
Push Manager	4
Push Submitter 1	2
Push Submitter 2	3
Push Submitter 3	8
Push Submitter 4	10
Push Submitter 5	unsure
Push Submitter 6	5
Push Submitter 7	0

4. Since the implementation of the code deployment tool, on average, approximately how many times per month have there been SVN conflicts that ended up affecting production systems?

Table 4-5 Survey Question #4 Data

Respondent	Answer
Push Manager	0
Push Submitter 1	0
Push Submitter 2	1
Push Submitter 3	1
Push Submitter 4	2
Push Submitter 5	unsure
Push Submitter 6	1
Push Submitter 7	0

5. Prior to the use of the code deployment tool, on average, approximately how many times per month was code unintentionally pushed to production systems?

Table 4-6 Survey Question #6 Data

Respondent	Answer
Push Manager	6
Push Submitter 1	many
Push Submitter 2	1
Push Submitter 3	5
Push Submitter 4	3
Push Submitter 5	3
Push Submitter 6	unsure
Push Submitter 7	0

6. Since the implementation of the code deployment tool, on average, approximately how many times per month has code unintentionally been pushed to production systems?

Table 4-7 Survey Question #6 Data

Respondent	Answer
Push Manager	0
Push Submitter 1	0
Push Submitter 2	0.5
Push Submitter 3	0
Push Submitter 4	1
Push Submitter 5	0
Push Submitter 6	unsure
Push Submitter 7	0

7. Prior to the use of the code deployment tool, on average, approximately how many times per month were there SQL related issues on production systems?

Table 4-8 Survey Question #7 Data

Respondent	Answer
Push Manager	4
Push Submitter 1	unsure
Push Submitter 2	unsure
Push Submitter 3	3
Push Submitter 4	many
Push Submitter 5	unsure
Push Submitter 6	unsure
Push Submitter 7	unsure

8. Since the implementation of the code deployment tool, on average, approximately how many times per month have there been SQL related issues on production systems?

Table 4-9 Survey Question #8 Data

Respondent	Answer
Push Manager	0
Push Submitter 1	unsure
Push Submitter 2	unsure
Push Submitter 3	0
Push Submitter 4	1
Push Submitter 5	unsure
Push Submitter 6	unsure
Push Submitter 7	unsure

9. Prior to the use of the code deployment tool, on average, on a normal push day, how many minutes did you spend submitting, validating, and processing push requests?

Table 4-10 Survey Question #9 Data

Respondent	Answer
Push Manager	120
Push Submitter 1	120
Push Submitter 2	8
Push Submitter 3	15
Push Submitter 4	120
Push Submitter 5	many
Push Submitter 6	45
Push Submitter 7	20

10. Since the implementation of the code deployment tool, on average, on a normal push day, how many minutes do you spend submitting, validating, and processing push requests?

Table 4-11 Survey Question #10 Data

Respondent	Answer
Push Manager	30
Push Submitter 1	60
Push Submitter 2	10
Push Submitter 3	10
Push Submitter 4	90
Push Submitter 5	many
Push Submitter 6	10
Push Submitter 7	10

11. Do you think the implementation of the code deployment tool has made the Doba deployment process more RELIABLE? Rate this on a scale from ONE to SEVEN where ONE is much more reliable, FOUR is just as reliable and SEVEN is less reliable. Please explain your answer.

Table 4-12 Survey Question #11 Data

Respondent	Answer
Push Manager	1
Push Submitter 1	unsure
Push Submitter 2	2
Push Submitter 3	4
Push Submitter 4	4
Push Submitter 5	3
Push Submitter 6	2
Push Submitter 7	1

12. Do you think the implementation of the code deployment tool has made the Doba deployment process more STABLE? Rate this on a scale from ONE to SEVEN where ONE is much more stable, FOUR is just as stable and SEVEN is less stable. Please explain your answer.

Table 4-13 Survey Question #12 Data

Respondent	Answer
Push Manager	1
Push Submitter 1	unsure
Push Submitter 2	2
Push Submitter 3	2
Push Submitter 4	2
Push Submitter 5	3
Push Submitter 6	2
Push Submitter 7	1

13. Do you think the implementation of the code deployment tool has made the Doba deployment process more ACCURATE? Rate this on a scale from ONE to SEVEN where ONE is much more accurate, FOUR is just as accurate and SEVEN is less accurate. Please explain your answer.

Table 4-14 Survey Question #13 Data

Respondent	Answer
Push Manager	1
Push Submitter 1	unsure
Push Submitter 2	2
Push Submitter 3	1
Push Submitter 4	2
Push Submitter 5	3
Push Submitter 6	2
Push Submitter 7	5

14. Since the implementation of the code deployment tool, do you think the testing (staging) environment is more or less effective? Rate this on a scale from ONE to SEVEN where ONE is much more effective, FOUR is just as effective and SEVEN is less effective. Please explain your answer.

Table 4-15 Survey Question #14 Data

Respondent	Answer
Push Manager	1
Push Submitter 1	2
Push Submitter 2	4
Push Submitter 3	1
Push Submitter 4	2
Push Submitter 5	2
Push Submitter 6	2
Push Submitter 7	4

15. Are there any improvements you have seen made through the use of the code deployment tool not previously mentioned in this survey? If so, please list them.
    - *Push requests are much simpler to make*
    - *Rolling back code is a lot easier*
    - *Reviewing past pushes is easier*

16. Are there any drawbacks you have seen introduced by the code deployment tool that have not been previously mentioned in this survey? If so, please list them.
    - *Manual actions are more tedious now.*
    - *There is no synchronous release of externals, so when one is pushed, there is a delay while the other one is pushed where unmet dependencies exist.*
    - *You can't push code from two projects in the same push request. No longer an email of all changes pushed to production.*
    - *Would like to see user allowed to push own code.*
    - *Would like to have tool automatically create new testing branch after push.*

4.2.5.3 Survey Results – Analyzed

In order to analyze the results, an average of the recorded answers was calculated. For questions dealing with differences between the old process and the new research method (numbers 3 and 4, 5 and 6, 7 and 8, 9 and 10), a difference was taken for each user and the percentage of change averaged across all users. The following are the results of the analysis of the survey responses.

4.2.5.3.1 Percent Change In SVN Conflicts

In order to determine the impact of the new system on improving the occurrence of SVN conflicts on production systems, the data for survey questions 3 and 4 were analyzed. The percentage change between the two for each user was computed and then averaged to show an overall decrease in SVN conflicts of 74%.

Since theoretically, it is not possible for SVN conflicts to occur in production (if there is a conflict in the merge, the changes will not be pushed), it appears, according to the accompanying comments, that the respondents who indicated a positive number of conflicts for the post-deployment tool time period were actually referring to conflicts when pushing to the testing environment, which occur by design, most commonly when multiple dependent revisions are committed and then not included in the push request. In reality this number should be at -100%, but the survey results do indicate that users are more confident in and pleased with the decrease in SVN conflicts. The table below describes how these conclusions were reached.

Table 4-16 Average Change In SVN Conflicts

Respondent	Pre-Research (#3)	Post-Research (#4)	% Difference
Push Manager	4	0	-100%
Push Submitter 1	2	0	-100%
Push Submitter 2	3	1	-67%
Push Submitter 3	8	1	-88%
Push Submitter 4	10	2	-80%
Push Submitter 5	unsure	unsure	n/a
Push Submitter 6	5	1	-80%
Push Submitter 7	0	0	0%
Average			-74%

4.2.5.4 Percent Change In Unintended Pushes

Before being able to push by specific revision number, it was common for previously committed revisions to be pushed unintentionally with code dependent on revisions of other files that may or may not also be pushed. By allowing and requiring push requests to be tied to specific revision numbers, this problem should be alleviated. Using the data from survey questions #5 and #6, the percentage change for each respondent was calculated before and after the use of the deployment tool, and then the percentages of all respondents were averaged. This yields a decrease in unintended pushes of 74%. Since these can still occur due to user error, this is a reasonable value.

Table 4-17 Average Change In Unintended Pushes

Respondent	Pre-Research (#5)	Post-Research (#6)	% Difference
Push Manager	6	0	-100%
Push Submitter 1	many	0	-100%
Push Submitter 2	1	0.5	-50%
Push Submitter 3	5	0	-100%
Push Submitter 4	3	1	-67%
Push Submitter 5	3	0	-100%
Push Submitter 6	unsure	unsure	n/a
Push Submitter 7	0	0	0%
Average			-74%

4.2.5.5 Percent Change In SQL Issues

In order to determine how much improvement the new system has made in preventing and resolving SQL issues during the push process, the data from survey questions #7 and #8 were used. A percentage difference was calculated for each respondent and then those percentages were averaged. According to the results, the prevalence of SQL issues has declined by 92% with the introduction of the code deployment tool, which is a reasonable and expected result. Since a fair number of respondents surveyed were unsure on these questions, the results are only based on a few answers, but still fall within the expected range.

Table 4-18 Average Change In SQL Issues

Respondent	Pre-Research (#7)	Post-Research (#8)	% Difference
Push Manager	4	0	-100%
Push Submitter 1	unsure	unsure	n/a
Push Submitter 2	unsure	unsure	n/a
Push Submitter 3	3	0	-100%
Push Submitter 4	4	1	-75.00%
Push Submitter 5	unsure	unsure	n/a
Push Submitter 6	unsure	unsure	n/a
Push Submitter 7	unsure	unsure	n/a
Average			-92%

4.2.5.6 Percent Change In Push Time

One of the goals of the research was to reduce the amount of time needed to both submit push requests and process them. To analyze whether this was accomplished, the data from survey questions #9 and #10 were used. For each respondent, the percent difference was calculated, and was then averaged with the percent change for all other respondents. It was determined from this method that the average change in push time was a decrease of 41%.

Table 4-19 Average Change In Push Time

Respondent	Pre-Research (#9)	Post-Research (#10)	% Difference
Push Manager	120 min	30 min	-75%
Push Submitter 1	120 min	60 min	-50%
Push Submitter 2	8 min	10 min	25%
Push Submitter 3	15 min	10 min	-33%
Push Submitter 4	120 min	90 min	-25%
Push Submitter 5	many	many	n/a
Push Submitter 6	45 min	10 min	-78%
Push Submitter 7	20 min	10 min	-50%
Average			-41%

4.2.5.7 Average Reliability

Using the data from survey question #11, where users are asked to rate the increase in push process reliability since the introduction of the code deployment tool, a reliability rating has been established. Respondents were asked to rate reliability improvement on a scale from one to seven, where one is much more reliable, four is just as reliable and seven is less reliable. The average of their responses equates to a reliability rating of 2.4, as described below, indicating a marked improvement in deployment reliability.

Table 4-20 Average Reliability Rating

Respondent	Reliability
Push Manager	1
Push Submitter 1	unsure
Push Submitter 2	2
Push Submitter 3	4
Push Submitter 4	4
Push Submitter 5	3
Push Submitter 6	2
Push Submitter 7	1
Average	2.4

4.2.5.8 Average Stability

Survey question #12 provides responses indicating the increased stability provided by the code deployment tool. This uses the same scale from one to seven as was used to determine reliability. According to the survey data, a stability rating of 1.9 was determined by averaging the responses, indicating a significant increase in stability.

Table 4-21 Average Stability Rating

Respondent	Stability
Push Manager	1
Push Submitter 1	unsure
Push Submitter 2	2
Push Submitter 3	2
Push Submitter 4	2
Push Submitter 5	3
Push Submitter 6	2
Push Submitter 7	1
Average	1.9

4.2.5.9 Average Accuracy

The accuracy of the new code deployment process was determined using the averaged results of the responses to survey question #13, which again used the same scale from one to seven as the previous ratings. Averaging these results yields an accuracy rating of 2.2, which again indicates significant improvement in this category.

Table 4-22 Average Accuracy Rating

Respondent	Accuracy
Push Manager	1
Push Submitter 1	unsure
Push Submitter 2	2
Push Submitter 3	1
Push Submitter 4	2
Push Submitter 5	3
Push Submitter 6	2
Push Submitter 7	5
Average	2.2

4.2.5.10Staging Environment Improvement

According to the results of survey question #14, the code deployment tool has also provided increased improvement in the staging environment. According to the averaged results, the improvement rating for the staging environment is 2.3. This is on the same scale from one to seven as the previous ratings, and therefore shows a significant increase in the effectiveness of the staging environment.

Table 4-23 Average Staging Environment Improvement Rating

Respondent	Improvement
Push Manager	1
Push Submitter 1	2
Push Submitter 2	4
Push Submitter 3	1
Push Submitter 4	2
Push Submitter 5	2
Push Submitter 6	2
Push Submitter 7	4
Average	2.3

4.2.6 Repository Quality

One critical component to the effectiveness of the code deployment tool that was discovered through this research is the importance of starting out with final destination branches, and subsequently testing branches, that closely match code in their project's trunk. Since merges are used to push code from the trunk to testing branches and then from the testing branches to their final destination branches, discrepancies in the trunk and those branches can lead to issues when merging.

In the case of the Doba production branch, when the code deployment tool was first employed as part of the Doba deployment process, a production branch needed to be created. Since the previous push process did nothing more than update select files in a working copy on the production systems, there was no real way to determine which versions of each file were being used to be able to create a branch without significant manual work. Therefore, a production branch was created directly from the

production working copy. One implication of this method, and the fact that there was a lack of discretion when making changes directly on production, is that the production branch contained local modifications that had never been committed to the trunk. The result of this were conflicts when pushing those files from the trunk when the changes that were made locally were very similar to those being pushed. Usually this occurred when a change that existed independently in the branch was made in the trunk and is then pushed into the branch. Since the change already existed, a merge conflict occurred.

Another implication of this method of creating the original production branch is that many files were deleted from the trunk, but those changes were never pushed to the production working copy. Therefore, when doing a merge of the trunk into the destination branch, like what is done for the folder containing a file that is being pushed *up_to_revision*, there was often a significant number of files that must be deleted as part of the merge (those files that were deleted from the trunk, which is the source branch, but never pushed to the target of the merge, the production branch), thereby costing performance.

Both of these issues can be resolved by synchronizing the trunk with the final destination branch (from which testing branches are created). This can be accomplished in one of two foreseeable ways. The first is to lock down the trunk (prevent anyone from committing code), and then merge all its changes into the final destination branch. This has the somewhat significant danger of pushing unknown changes existing in the trunk into the final destination branch.

The preferred method for fixing discrepancies between the trunk and the final destination (production) branch, is to lock down the trunk and the create a new trunk by copying the production branch. Any folder can easily be copied using *svn copy*, while still preserving the versioning history within the copy. This ensures that the production code base is not altered while synchronizing it with trunk.

Once the trunk has been synchronized with the final destination branch, the only remaining issue is that the two may drift apart over time. This occurs when changes are committed to the trunk but never pushed to the production branch. This problem was not addressed in the research, but would involve some mechanism to either periodically synchronize the two using one of the methods above, or to periodically remove changes to the trunk that have remained unpushed for some designated period of time.

5 Conclusions and Recommendations

This chapter discusses a summary of the conclusions arrived at after completing the research and implementing the code deployment tool and also offers some recommendations for future research.

5.1 Research Summary

By implementing a web-based resource management system that leverages the power of versioning software and a relational database management platform, the efficiency of resource deployment and test environment management in a multi-tiered web environment can be greatly increased, while also vastly improving the manageability and effective deployment of specific versions of the web-based application. This is evident in the decrease in SVN conflicts affecting production (reduced by 74%), the reduced number of incidents where code was errantly pushed (a decrease of 74%), the number of times that SQL issues occur on production (decreased by 92%), and the reduction in time required to submit and process push requests (decreased by 41%). Furthermore, the research has increased the ability of developers to track down their own issues using the log of deployed resources, and they have rated it as more reliable, more stable, and more accurate. Users also indicate that the staging environment is now more reflective of production.

Built using a LAMP (Linux, Apache, MySQL, and PHP) set up to manage source code, binary, database, and cron scripted resources, this software provides a completely open source solution for web based resource management (LAMP, 2007). A central server hosts the deployment tool and stores its configurations and push request settings in a dedicated database so that it can interact with the affected resources to effectively deploy version changes to them in a timely and efficient manner. The system is also scalable to allow the management of additional resources as the target system's architecture grows.

The deployment tool involved in this research relies heavily on the branching abilities of the Subversion code versioning platform to provide separate and easily compartmentalized versions of the source code and binary resources being managed by the system. This allows any configured version of the managed application to be checked out into an independent testing environment and validated before being released to its final production destination. The inclusion of this business logic process effectively allows a quality assurance and management approval element to the push process.

Resources that can be managed by the code deployment tool are any versionable source code or binary elements, cron files, database resources, and executed scripts or other commands. The deployment tool not only provides a much more efficient way to manage the deployment of these resources, but also creates an architecture that makes it much easier to roll back resources pushed in the event of catastrophe. Since updates to a production level push branch occurs in a single commit (when a push branch

is merged into its final destination), this revision can easily be rolled back to the previous revision in the case of erroneous code being pushed. Since all necessary resources can be pushed through the deployment tool, a policy can be employed that changes are only pushed through the tool, thereby creating an accurate log of all changes that occur.

5.2 Conclusions

The following is a list of conclusions that have been garnered from this research:

- Creating a web based interface for the deployment tool has proved valuable as it allows easy access for the original developer of content, or even a QA engineer approving it, to provide the specifications on what should be pushed. This is beneficial, since these are the individuals who know best what resources are needed.
- Building the system on open source technology (PHP, Smarty, Subversion, MySQL, etc.) allows the research to be adopted more widely and freely accepted by potential users as there are no financially limiting license restrictions. The source code used in this research has been released to the open source community under the GPL and can be found hosted at http://sourceforge.net/projects/deploid/.
- Since it has been designed from inception with security in mind, when properly installed, the deployment tool has been created with checks in place to prevent SQL injection, brute force login cracking, database cracking, man-in-the-middle attacks, and the

sniffing of sensitive data from a shared network. It has also adopted an architecture that does not require authentication information for all production resources to be stored in a single place, therefore a breach of the deployment tool system will not also intrinsically yield complete control over each resource being managed.

- The introduction of quality assurance and validation mechanisms into the deployment tool allows it to be more fully integrated with the business logic dictated by a higher level work flow process. For example, the tool can be used to allow QA players or managers to hold off a push from being processed until all their needs have been met.

- Since the deployment tool integrates intimately with Subversion by dictating a repository's layout, it provides a valuable model for organizing a project's layout in Subversion that can be beneficial even outside the scope of the deployment tool. By dictating a separate branch for each notable version of the application it allows Subversion to be utilized independently from the deployment tool to test and push source code.

- In order to test its effectiveness, the deployment tool has been utilized to manage the testing and production resources of a real world entity, http://www.doba.com. This has succeeded in providing greater stability, faster deployment, more accountability, more secure systems, and more efficient resource utilization than what existed prior to the implementation of the research. According to the results of a post usage survey conducted with system users after employing

the deployment tool within the Doba use case, SVN conflicts on production have been reduced by 74%, unintended pushes have been reduced by 74%, SQL issues affecting production have been reduced by 92%, and the time spent by users doing push requests has been reduced by 41%. In addition, on a scale from one to seven where one is much more reliable, four is just as reliable and seven is less reliable when comparing the research to the previous deployment process, respondents rated reliability at 2.4, stability at 1.9, accuracy at 2.2, and an improvement of the staging environment at 2.3, showing substantial improvement in each of these categories.

- The following concerns still remain among users of the tool according to the results of the post usage survey:

    1. Manual actions have now become more tedious. This refers mostly to the use of SVN branches, and the manual merging and manipulation that sometimes must be done to troubleshoot issues.

    2. When releasing two or more branches that are externally linked, there is a dependency that is not accounted for. Therefore, there is a period of time when one branch has been pushed ans the other is either being pushed or is waiting, that the dependency is not met, potentially breaking the code momentarily, until all branches have been successfully pushed.

    3. You can no longer push code from multiple projects in the same push request, making it a bit more inconvenient.

4. It would be an added convenience to allow the tool to automatically create a new testing branch when the current one is pushed.

5.3 Recommendations for Future Research

One important area of research that will increase the value of the deployment tool system is the development of plug-ins and support for additional database and versioning platforms. Currently the system only allows for the use of Subversion and MySQL to accomplish these tasks, however as mentioned previously there are a multitude of other platforms being utilized in the web application arena.

Another area of research that did not fit within the scope of this project is that of database versioning. In general, it seems easiest to allow this to be managed at the database application level, but it could theoretically be integrated into the deployment tool itself. If it were to be managed by the database application, this could be accomplished by simply providing support for acceptable platforms and integrating their functionality into the tool.

The existing implementations of QA and management interaction in the deployment process are fairly basic (they essentially only provide approval or disapproval) and could be expanded to create more interaction. This could include the integration of interaction with defect tracking and project management software.

Another area of research pertinent to the usage of the deployment tool is further automation of the creation of testing environments. Since the

tool manages versions of an application, which must currently be manually configured (the source code checkout, database configuration, etc) it would be beneficial to further automate this process to allow specific versions to be quickly tested in an isolated environment. The ideal solution would be to utilize a virtual machine architecture in which virtual machines can be systematically created and destroyed with ease.

As was discussed in chapter 4, there remains a need to be able to periodically synchronize the final destination branch with its project trunk. Probably the most ideal solution to this issue would be to have some process as part of the code deployment tool that periodically (perhaps weekly) compares the trunk with the production branch to determine the differences. This could keep track of how long changes remain unpushed in the trunk and allow them to either be pushed or be removed if they will never end up being pushed.

Finally, in order to appeal to the largest number of users, the deployment tool should be tested and validated for use with other major web browsers. Its current support only in Firefox limits it to users who have Firefox available, and making it cross-browser compatible will increase its appeal.

6 References

@Task Project Management Software. 2007. Online. Available from Internet, http://www.attask.com/, accessed 14 June 2007.

Alexander, C. 1999. The Origin of Pattern Theory. *IEEE Software* 48 (September/October): 71.

Bright Lemon. 2007. *Info About Database Platforms*. Online. Available from Internet, http://www.brightlemon.com/web-design/resources/database-platforms.php, accessed 16 April 2007.

Co-hosting Multiple Versions of J2EE Applications. 2004. Online. Available from Internet, http://www.ibm.com/developerworks/websphere/techjournal/0405_poddar/0405_poddar.html, accessed 24 Oct 2007.

Codd, E. 1970. *A Relational Model of Data For Large Shared Data Banks,* Reprinted from *Communications of the ACM*, Vol. 13, No. 6, June 1970, pp. 377-387. Online. Available from Internet, http://www.acm.org/classics/nov95/toc.html, accessed 16 April 2007.

Dickerson, M. 2007. Personal communication with Bryce Ott. 5 April.

Doba. 2007. Online. Available from Internet, http://www.doba.com, accessed 3 June 2007.

Fayad, M. and Schmidt, D. 1997. Object Oriented Application Frameworks. *Communications of the ACM* 40: 32.

Grune, D. 2004. *Concurrent Versions System CVS.* Online. Available from Internet, http://www.cs.vu.nl/~dick/CVS.html, accessed 18 June 2007.

Helman, D. 1998. *Model-View-Controller*. Online. Available from Internet, http://ootips.org/mvc-pattern.html, accessed 18 June 2007.

Hunt, J. and Reuter J. 2001. Using the Web for Documented Versioning: An Implementation Report for DeltaV. *IEEE Archive:* 508.

Java Solutions. 2007. Online. Available from Internet, http://www.parasoft.com/jsp/solutions/home.jsp?solution=JavaT&itemId=178, accessed 24 Oct 2007.

J2EE Packaging and Development. 2007. *Professional Java Server Programming J2EE 1.3 Edition.* Online. Available from Internet,

http://www.theserverside.com/tt/articles/content/J2EE-Deployment/chapter.html, accessed 24 Oct 2007.

LAMP. 2007. Online. Available from Internet, http://www.onlamp.com/, accessed 16 Sep 2007.

Little Tech Shoppe. 1994. *Crontab Man Page*. Online. Available from Internet, http://www.littletechshoppe.com/servers/extensions/cron/crontab_5.html,, accessed 16 April 2007.

Marr, S. 2005. *The Java Data Objects Persistence Model*. Online. Available from Internet, http://www.stefan-marr.de/artikel/jdo-persistence-model/paper.html, accessed 16 April 2007.

McIlroy, M. 1976. Mass-produced Software Components. *Software Engineering Concepts and Techniques, Proceedings of 1968 North Atlantic Treaty Organization (NATO) Conference on Software Engineering Garmisch-Partenkirchen*: 88.

Microsoft Office SharePoint Server 2007 Evaluation Guide. 2007. Online. Available from Internet, http://office.microsoft.com/search/redir.aspx?AssetID=XT101662731033&CTT=5&Origin=HA101680161033, accessed 24 Oct 2007.

Mohlman, T. and Jacobs, J. 2007. Personal communication with Bryce Ott. 30 March.

Moreira, V. and Edelweiss, N. 1999. Schema Versioning: Queries to The Generalized Temporal Database System. *Tenth International Workshop on Database and Expert Systems Applications* (September): 458-459.

MySQL AB. 2007. *The world's most popular open source database.* Online. Available from Internet, http://mysql.com/, accessed 16 Sep 2007.

Netcraft. 2007. *March 2007 Web Server Survey*. Online. Available from Internet, http://news.netcraft.com/archives/2007/02/23/march_2007_web_server_survey.html, accessed 15 April 2007.

NonGNU. 2007. *Concurrent Versions System.* Online. Available from Internet, http://www.nongnu.org/cvs/, accessed 17 April 2007.

Online CVS Manual. 2007. *Overview.* Online. Available from Internet, http://ximbiot.com/cvs/manual/cvs-1.11.22/cvs_1.html#SEC1, accessed 17 April 2007.

Ott, B. 2007. *Deploid*. Online. Available from Internet, http://sourceforge.net/projects/deploid/, accessed 3 June 2007.

Parnas, D. 1972. On the Criteria To Be Used in Decomposing Systems into Modules. *Communications of the ACM* 15 (December): 1053.

PHP: Hypertext Processor. 2007. Online. Available from Internet, http://php.net/, accessed 16 Sep 2007.

Roddick, J. 1996. A Survey of Schema Versioning Issues for Database Systems. *Information and Software Technology* 37: 383-393.

Schlossnagle, T. 2006. *Scalable Internet Architectures.* Indianapolis: Sams Publishing.

Smarty: Template Engine. 2007.Online. Available from Internet, http://smarty.php.net/, accessed 16 Sep 2007.

Speck, A. and Pulvermuller, E. 2001. Versioning in Software Engineering. *The 27th Annual Conference of the IEEE Industrial Electronics Society*: 1856.

Subversion. 2007. *Subversion Features.* Online. Available from Internet, http://subversion.tigris.org/, accessed 17 April 2007.

Subversion Book. 2007. *Version Control With Subversion for Subversion 1.4, compiled from r2782.* Online. Available from Internet, http://svnbook.red-bean.com/nightly/en/svn-book.html, accessed 17 April 2007.

Szyperski, C. 1997. *Component Software.* New York: Addison-Wesley.

Vignette Content Management. 2007. Online. Available from Internet, http://www.vignette.com/portal/site/us/menuitem.dcbb524431151aaa32189210180141a0/?vgnextoid=86a295338521b010VgnVCM1000005610140aRCRD&vgnext-selected-menuitem=4b09bdd80b8ff1e8fb3d8010180141a0, accessed 24 Oct 2007.

Web Services Solutions. 2007. Online. Available from Internet, http://www.parasoft.com/jsp/solutions/home.jsp?solution=WebServT, accessed 24 Oct 2007.

Wei, H. and Elmasri, R. 1999. Study and Comparison of Schema Versioning and Database Conversion Techniques for Bi-Temporal Databases. *Proceedings of Sixth International Workshop on Temporal Representation and Reasoning* (May): 88-98.

Wei, H. and Elmasri, R. 2000. PMTV: A Schema Versioning Approach for Bi-Temporal Databases. *Proceedings Seventh International Workshop on Temporal Representation and Reasoning* (July): 143-151.

Whitehead, E. and Goland, Y. 2007. *WebDAV, A Remote Protocol for Remote Collaborative Authoring on the Web*. Online. Available from Internet, http://www.ics.uci.edu/~ejw/papers/dav-ecscw.pdf, accessed 17 April 2007.

Whitehead, E. and Wiggins, M. 1998. WEBDAV: IETF Standard for Collaborative Authoring on the Web. *IEEE Internet Computing* (September/October): 34.

Wikipedia. 2007. *Database. Online*. Available from Internet, http://en.wikipedia.org/wiki/Database, accessed 16 April 2007.

Wikipedia. 2007. *Revision Control.* Online. Available from Internet, http://en.wikipedia.org/wiki/Versioning, accessed 17 April 2007.

Yahoo! UI Library. 2007. Online. Available from Internet, http://developer.yahoo.com/yui/, accessed 21 May 2007.

Appendices

Appendix A: Database Schema

This appendix contains details on the definition of the database schema used by the deployment tool for managing resources and system configurations.

A.1 Schema Diagram

The following is a schema diagram showing the normalized database used by the deployment tool. It details the attributes present in each table as well as the relationships which exist between entities within the schema.

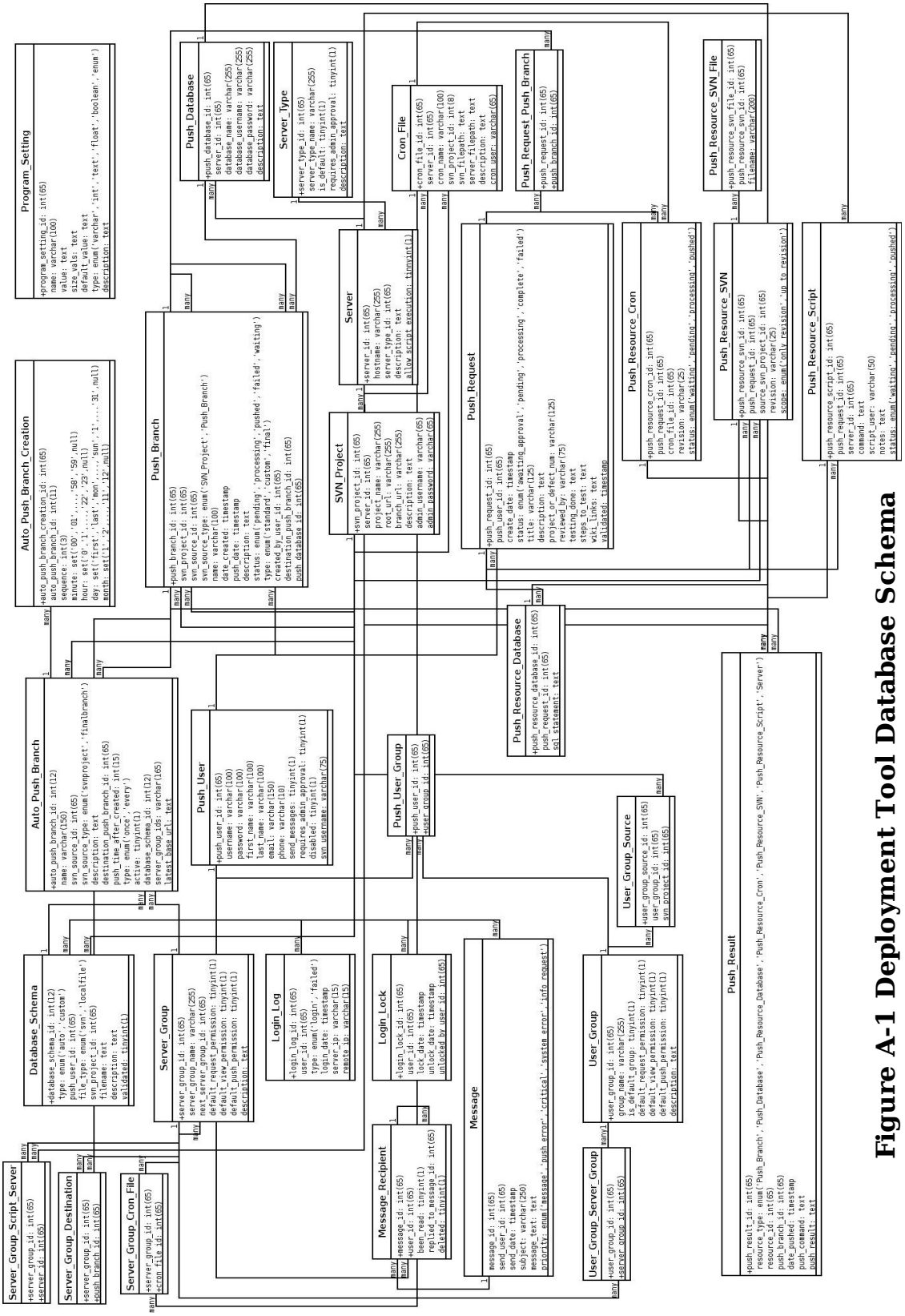

Figure A-1 Deployment Tool Database Schema

A.2 Schema Table Descriptions

The following is a list of each table in the database schema along with a brief description of its purpose.

A.2.1 Auto_Push_Branch

This table stores information regarding the creation of automated testing push branches, including the source for the branch, its final destination branch, and information about any attached database.

A.2.2 Auto_Push_Branch_Creation

Working in conjunction with the Auto_Push_Branch table, this table stores information on the scheduled frequency of when an automated testing branch should be created. It defines the minutes, hours, days, and months of the scheduled creation.

A.2.3 Cron_File

This table stores information regarding cron files that are managed through the deployment tool. Information stored here includes the server where the cron will be run, the user it should run as, and the location of the versioned source for the cron file.

A.2.4 Database_Schema

In order to create new databases for use with automatically created test branches, this table stores information about possible database schema files. Those files can be either SQL files uploaded onto the deployment tool

server, or files versioned in SVN, as defined by the *file_type* attribute. Records in this table define the location of the schema file, the user that created it, and whether or not it has been validated as safe or not.

A.2.5 Login_Lock

Entries in this table indicate that a user account has been locked out for too many failed login attempts. Records contain information on the user locked out, when the lock was created, when it was unlocked and by whom (if it has been unlocked).

A.2.6 Login_Log

Each time a valid username is used to attempt login, an entry is created for it here in this table. Also stored is whether the login failed or was successful, what time it occurred, and the IP address of the client requesting login.

A.2.7 Message

This table contains records that hold data regarding messages that have been sent in the system, either by users or for notification of system events. It stores who sent the message, its subject and contents, the date it was sent, and the message priority. No matter how many users the message was sent to, its contents are only stored here once.

A.2.8 Message_Recipient

Entries in this table refer to recipients of messages that are stored in the *Message* table. It is normalized to prevent the same message sent to multiple users from being duplicated.

A.2.9 Program_Setting

The contents of this table are the values of global system settings for the deployment tool. It is designed to be generic such that any type of global setting can be stored here. It defines the data type of the setting and provides a name value pairing for it.

A.2.10 Push_Branch

This table stores information about push branches that are being managed by the system. As defined by the *type* attribute, these branches can be custom created, automatically generated, or be final push branches. Final push branches cannot be merged into another branch and are treated as the final destination for other testing branches. This table also defines whether the push branch is tied to a database resource for processing SQL changes.

A.2.11 Push_Database

The Push_Database table is used to store information about database resources that are being managed by the deployment tool system. Included in these records is information about the server hosting the database as well as authentication information needed to connect to it.

A.2.12 Push_Request

This table stores information about push requests that have been added to the system, including the user that submitted them, the status of the push request, other details about it, and whether or not it has been validated or not.

A.2.13 Push_Request_Push_Branch

This table is used merely to tie push requests to the destination push branches where they are to be pushed. It contains only the push request id and destination push branch id for the resources involved.

A.2.14 Push_Resource_Cron

Records in this table refer to cron file changes that have been requested as part of a push request, including the base cron file, what revision to push, and the status of the change.

A.2.15 Push_Resource_Database

This table holds SQL changes that have been added as part of a push request. It defines merely the SQL change to be enacted and the push request it belongs to.

A.2.16 Push_Resource_SVN

As part of a push request, users can request that SVN versioned resources be pushed, This table stores the source for those resources, what revision to push, and also the scope of the push. The scope defines whether

the just the changes made for that revision should be pushed, or all changes leading up to and including that revision. The actual files to be pushed from the specified revision are defined in the *Push_Resource_SVN_File* table.

A.2.17 Push_Resource_SVN_File

The table defines which files from a particular revision (as defined by the corresponding entry from *Push_Resource_SVN*) should be pushed as part of a push request. It only contains the id of the corresponding *Push_Resource_SVN* entry and the filename.

A.2.18 Push_Resource_Script

Scripts can be run as part of a push request, and entries in this table define how that type of change should be processed. This includes the command to be run, the server to run it on, which user it should be run as, and the status of the request.

A.2.19 Push_Result

Records in this table correspond to the results of pushes being processed. According to the *resource_type* attribute, this can be from any of the four resources that can be pushed in a push request (SVN file, SQL change, cron file change, or script), from a push branch being merged into its destination, the database from a push branch merged into its destination, or the results of the update to a working copy on a destination server. The command as well as the results are stored in this table.

A.2.20 Push_User

This table stores information about users of the deployment tool system. Username, password, contact information, SVN username, and default permissions are all stored here. Passwords are encrypted using the AES encryption functionality of MySQL.

A.2.21 Push_User_Group

This table contains records that tie a push user to a specific user group and contains only the user id and the corresponding user group id.

A.2.22 SVN Project

Records here are generated from the details of SVN projects managed by the deployment tool. These are projects associated with the SVN source for SVN related activities. Contained in each record is the root URL of the SVN project as well as the URL for storing branches from that project and the necessary authentication credentials to access resources from the project.

A.2.23 Server

This table stores information about servers with resources being managed by the deployment tool. Records define the server hostname, its primary type and whether or not it allows scripts to be executed on it.

A.2..24 Server_Group

This table stores information on server groups that are used to define the permissions on a group of server resources in a broader sense. It defines the permissions for the group, along with a group name and description.

A.2.25 Server_Group_Cron_File

This table defines cron file objects that are members of a specific server group, and contains only the server group and cron file ids.

A.2.26 Server_Group_Destination

This table defines which push branches can be defined as final destination branches (production branches) for the specified server group. It contains only the server group id and push branch id.

A.2.27 Server_Group_Script_Server

Records in this table indicate which script servers fall within the scope of the server group. It contains only the server group and server ids.

A.2.28 Server_Type

This table contains the types of servers that are available in the system. Most of these are default, as defined by the *is_default* attribute, but this table exists to allow custom server types to be added in the future to expand functionality.

A.2.29 User_Group

To better manage permissions in a larger group of users, this table stores information about user groups. This information includes a name, description, and applicable permissions.

A.2.30 User_Group_Server_Group

The purpose of this table is to tie user groups to server groups to give user groups permissions to access server groups. It contains only the user group and server group ids.

A.2.31 User_Group_Source

This table defines which SVN sources are available to members of a particular user group. It contains the user group id and SVN project id that the source comes from.

A.3 Schema SQL

The following is the raw SQL output for the MySQL database used to generate the deployment tool management database:

```
-- phpMyAdmin SQL Dump
-- version 2.6.0-rc1
-- http://www.phpmyadmin.net
--
-- Host: localhost
-- Generation Time: May 09, 2007 at 09:23 AM
-- Server version: 4.1.20
-- PHP Version: 4.3.9
--
-- Database: `bott_mastercodepush`
--

-- --------------------------------------------------------

--
-- Table structure for table `Auto_Push_Branch`
```

```
--

DROP TABLE IF EXISTS `Auto_Push_Branch`;
CREATE TABLE `Auto_Push_Branch` (
  `auto_push_branch_id` int(12) NOT NULL auto_increment,
  `name` varchar(150) NOT NULL default '',
  `svn_source_id` int(65) NOT NULL default '0',
  `svn_source_type` enum('svnproject','finalbranch') NOT NULL default 'svnproject',
  `description` text NOT NULL,
  `destination_push_branch_id` int(65) default NULL,
  `push_time_after_created` int(15) default NULL,
  `type` enum('once','every') NOT NULL default 'once',
  `active` tinyint(1) NOT NULL default '0',
  `database_schema_id` int(12) default NULL,
  `server_group_ids` varchar(165) default NULL,
  `latest_base_url` text,
  PRIMARY KEY  (`auto_push_branch_id`),
  KEY `name` (`name`),
  KEY `destination_push_branch_id` (`destination_push_branch_id`),
  KEY `active` (`active`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;

--
-- Dumping data for table `Auto_Push_Branch`
--


-- --------------------------------------------------------

--
-- Table structure for table `Auto_Push_Branch_Creation`
--

DROP TABLE IF EXISTS `Auto_Push_Branch_Creation`;
CREATE TABLE `Auto_Push_Branch_Creation` (
  `auto_push_branch_creation_id` int(65) NOT NULL auto_increment,
  `auto_push_branch_id` int(11) NOT NULL default '0',
  `sequence` int(3) NOT NULL default '0',
  `minute`
set('00','01','02','03','04','05','06','07','08','09','10','11','12','13','14','15','16',
'17','18','19','20','21','22','23','24','25','26','27','28','29','30','31','32','33','34'
,'35','36','37','38','39','40','41','42','43','44','45','46','47','48','49','50','51','52
','53','54','55','56','57','58','59','null') default NULL,
  `hour`
set('0','1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18',
'19','20','21','22','23','null') default NULL,
  `day`
set('first','last','mon','tue','wed','thu','fri','sat','sun','1','2','3','4','5','6','7',
'8','9','10','11','12','13','14','15','16','17','18','19','20','21','22','23','24','25','
26','27','28','29','30','31','null') default NULL,
  `month` set('1','2','3','4','5','6','7','8','9','10','11','12','null') default NULL,
  PRIMARY KEY  (`auto_push_branch_creation_id`),
  KEY `auto_push_branch_id` (`auto_push_branch_id`),
  KEY `auto_push_branch_id_2` (`auto_push_branch_id`,`sequence`),
  KEY `day` (`day`,`month`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;

--
-- Dumping data for table `Auto_Push_Branch_Creation`
--


-- --------------------------------------------------------

--
-- Table structure for table `Cron_File`
--

DROP TABLE IF EXISTS `Cron_File`;
```

```
CREATE TABLE `Cron_File` (
  `cron_file_id` int(65) NOT NULL auto_increment,
  `server_id` int(65) NOT NULL default '0',
  `cron_name` varchar(100) NOT NULL default '',
  `svn_project_id` int(8) NOT NULL default '0',
  `svn_filepath` text NOT NULL,
  `server_filepath` text NOT NULL,
  `description` text NOT NULL,
  `cron_user` varchar(65) NOT NULL default '',
  PRIMARY KEY  (`cron_file_id`),
  KEY `server_id` (`server_id`),
  KEY `cron_name` (`cron_name`),
  KEY `svn_project_id` (`svn_project_id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;

--
-- Dumping data for table `Cron_File`
--


-- --------------------------------------------------------

--
-- Table structure for table `Database_Schema`
--

DROP TABLE IF EXISTS `Database_Schema`;
CREATE TABLE `Database_Schema` (
  `database_schema_id` int(12) NOT NULL auto_increment,
  `type` enum('auto','custom') NOT NULL default 'auto',
  `push_user_id` int(65) NOT NULL default '0',
  `file_type` enum('svn','localfile') NOT NULL default 'svn',
  `svn_project_id` int(64) default NULL,
  `filename` text NOT NULL,
  `description` text NOT NULL,
  `validated` tinyint(1) NOT NULL default '0',
  PRIMARY KEY  (`database_schema_id`),
  KEY `type` (`type`),
  KEY `type_2` (`type`,`push_user_id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;

--
-- Dumping data for table `Database_Schema`
--


-- --------------------------------------------------------

--
-- Table structure for table `Message`
--

DROP TABLE IF EXISTS `Message`;
CREATE TABLE `Message` (
  `message_id` int(65) NOT NULL auto_increment,
  `send_user_id` int(65) NOT NULL default '0',
  `send_date` timestamp NOT NULL default '0000-00-00 00:00:00',
  `subject` varchar(250) NOT NULL default '',
  `message_text` text NOT NULL,
  `priority` enum('message','push_error','critical','system_error','info_request') NOT
NULL default 'message',
  PRIMARY KEY  (`message_id`),
  KEY `send_user_id` (`send_user_id`),
  KEY `send_date` (`send_date`),
  KEY `priority` (`priority`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=2 ;

--
-- Dumping data for table `Message`
```

```
--


-- --------------------------------------------------------

--
-- Table structure for table `Message_Recipient`
--

DROP TABLE IF EXISTS `Message_Recipient`;
CREATE TABLE `Message_Recipient` (
  `message_id` int(65) NOT NULL default '0',
  `user_id` int(65) NOT NULL default '0',
  `been_read` tinyint(4) NOT NULL default '0',
  `replied_to_message_id` tinyint(4) default NULL,
  `deleted` tinyint(4) NOT NULL default '0',
  PRIMARY KEY  (`message_id`,`user_id`),
  KEY `read` (`been_read`),
  KEY `replied_to_message_id` (`replied_to_message_id`),
  KEY `deleted` (`deleted`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--
-- Dumping data for table `Message_Recipient`
--


-- --------------------------------------------------------

--
-- Table structure for table `Program_Setting`
--

DROP TABLE IF EXISTS `Program_Setting`;
CREATE TABLE `Program_Setting` (
  `program_setting_id` int(65) NOT NULL auto_increment,
  `name` varchar(100) NOT NULL default '',
  `value` text NOT NULL,
  `size_vals` text NOT NULL,
  `default_value` text NOT NULL,
  `type` enum('varchar','int','text','float','boolean','enum') NOT NULL default
'varchar',
  `description` text NOT NULL,
  PRIMARY KEY  (`program_setting_id`),
  KEY `name` (`name`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=6 ;

--
-- Dumping data for table `Program_Setting`
--

INSERT INTO `Program_Setting` VALUES (1, 'Email Server Name', 'strongsad.doba.com',
'265', 'strongsad.doba.com', 'varchar', 'The email server for sending users messages from
the system.');
INSERT INTO `Program_Setting` VALUES (2, 'Email Server Port', '25', '10', '25', 'int',
'The port used by the mail server to send messages.');
INSERT INTO `Program_Setting` VALUES (3, 'Email Messages', 'true', '''true'',''false''',
'true', 'boolean', 'If true, all messages generated in the system will be emailed to
users by default.');
INSERT INTO `Program_Setting` VALUES (4, 'Admin Push Requests', 'false',
'''true'',''false''', 'false', 'varchar', 'If true, this allows members of the
Administrators group to process their own push requests.');
INSERT INTO `Program_Setting` VALUES (5, 'Number of Revisions to Show', '100', '10',
'100', 'int', 'The number of previous revision numbers to show when a user is selecting
files for a push request. This is used by the script that queries the repository for
changes committed by the current user.');

-- --------------------------------------------------------
```

```
--
-- Table structure for table `Push_Branch`
--

DROP TABLE IF EXISTS `Push_Branch`;
CREATE TABLE `Push_Branch` (
  `push_branch_id` int(65) NOT NULL auto_increment,
  `svn_project_id` int(65) NOT NULL default '0',
  `svn_source_id` int(65) NOT NULL default '0',
  `svn_source_type` enum('SVN_Project','Push_Branch') NOT NULL default 'SVN_Project',
  `name` varchar(100) NOT NULL default '',
  `date_created` timestamp NOT NULL default CURRENT_TIMESTAMP on update
CURRENT_TIMESTAMP,
  `push_date` timestamp NOT NULL default '0000-00-00 00:00:00',
  `description` text NOT NULL,
  `status` enum('pending','processing','pushed','failed','waiting') NOT NULL default
'pending',
  `type` enum('standard','custom','final') NOT NULL default 'standard',
  `created_by_user_id` int(65) NOT NULL default '0',
  `destination_push_branch_id` int(65) default NULL,
  `push_database_id` int(65) default NULL,
  PRIMARY KEY  (`push_branch_id`),
  KEY `destination_push_branch_id` (`destination_push_branch_id`),
  KEY `svn_project_id` (`svn_project_id`,`push_date`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;

--
-- Dumping data for table `Push_Branch`
--


-- --------------------------------------------------------

--
-- Table structure for table `Push_Database`
--

DROP TABLE IF EXISTS `Push_Database`;
CREATE TABLE `Push_Database` (
  `push_database_id` int(65) NOT NULL auto_increment,
  `server_id` int(65) NOT NULL default '0',
  `database_name` varchar(255) NOT NULL default '',
  `database_username` varchar(255) NOT NULL default '',
  `database_password` varchar(255) NOT NULL default '',
  `description` text NOT NULL,
  PRIMARY KEY  (`push_database_id`),
  KEY `server_id` (`server_id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;

--
-- Dumping data for table `Push_Database`
--


-- --------------------------------------------------------

--
-- Table structure for table `Push_Request`
--

DROP TABLE IF EXISTS `Push_Request`;
CREATE TABLE `Push_Request` (
  `push_request_id` int(65) NOT NULL auto_increment,
  `push_user_id` int(65) NOT NULL default '0',
  `create_date` timestamp NOT NULL default CURRENT_TIMESTAMP on update CURRENT_TIMESTAMP,
  `status` enum('awaiting_approval','pending','processing','complete','failed') NOT NULL
default 'pending',
  `title` varchar(125) NOT NULL default '',
  `description` text NOT NULL,
```

```
  `project_or_defect_num` varchar(125) NOT NULL default '',
  `reviewed_by` varchar(75) NOT NULL default '',
  `testing_done` text NOT NULL,
  `steps_to_test` text NOT NULL,
  `wiki_links` text,
  `push_before` varchar(125) default NULL,
  `push_after` varchar(125) default NULL,
  `schedule_before` timestamp NOT NULL default '0000-00-00 00:00:00',
  `schedule_after` timestamp NOT NULL default '0000-00-00 00:00:00',
  `validated` timestamp NOT NULL default '0000-00-00 00:00:00',
  PRIMARY KEY  (`push_request_id`),
  KEY `push_user_id` (`push_user_id`),
  KEY `create_date` (`create_date`),
  KEY `status` (`status`),
  KEY `schedule_before` (`schedule_before`),
  KEY `schedule_after` (`schedule_after`),
  KEY `validated` (`validated`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;

--
-- Dumping data for table `Push_Request`
--


-- --------------------------------------------------------

--
-- Table structure for table `Push_Request_Push_Branch`
--

DROP TABLE IF EXISTS `Push_Request_Push_Branch`;
CREATE TABLE `Push_Request_Push_Branch` (
  `push_request_id` int(65) NOT NULL default '0',
  `push_branch_id` int(65) NOT NULL default '0',
  PRIMARY KEY  (`push_request_id`,`push_branch_id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--
-- Dumping data for table `Push_Request_Push_Branch`
--


-- --------------------------------------------------------

--
-- Table structure for table `Push_Resource_Cron`
--

DROP TABLE IF EXISTS `Push_Resource_Cron`;
CREATE TABLE `Push_Resource_Cron` (
  `push_resource_cron_id` int(65) NOT NULL auto_increment,
  `push_request_id` int(65) NOT NULL default '0',
  `cron_file_id` int(65) NOT NULL default '0',
  `revision` varchar(25) NOT NULL default '',
  `status` enum('waiting','pending','processing','pushed') NOT NULL default 'waiting',
  PRIMARY KEY  (`push_resource_cron_id`),
  KEY `push_request_id` (`push_request_id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;

--
-- Dumping data for table `Push_Resource_Cron`
--


-- --------------------------------------------------------

--
-- Table structure for table `Push_Resource_Database`
--
```

```
DROP TABLE IF EXISTS `Push_Resource_Database`;
CREATE TABLE `Push_Resource_Database` (
  `push_resource_database_id` int(65) NOT NULL auto_increment,
  `push_request_id` int(65) NOT NULL default '0',
  `sql_statement` text NOT NULL,
  PRIMARY KEY  (`push_resource_database_id`),
  KEY `push_request_id` (`push_request_id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;

--
-- Dumping data for table `Push_Resource_Database`
--


-- --------------------------------------------------------

--
-- Table structure for table `Push_Resource_SVN`
--

DROP TABLE IF EXISTS `Push_Resource_SVN`;
CREATE TABLE `Push_Resource_SVN` (
  `push_resource_svn_id` int(65) NOT NULL auto_increment,
  `push_request_id` int(65) NOT NULL default '0',
  `source_svn_project_id` int(65) NOT NULL default '0',
  `revision` varchar(25) NOT NULL default '',
  `scope` enum('only_revision','up_to_revision') NOT NULL default 'only_revision',
  PRIMARY KEY  (`push_resource_svn_id`),
  KEY `push_request_id` (`push_request_id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;

--
-- Dumping data for table `Push_Resource_SVN`
--


-- --------------------------------------------------------

--
-- Table structure for table `Push_Resource_SVN_File`
--

DROP TABLE IF EXISTS `Push_Resource_SVN_File`;
CREATE TABLE `Push_Resource_SVN_File` (
  `push_resource_svn_file_id` int(65) NOT NULL auto_increment,
  `push_resource_svn_id` int(65) NOT NULL default '0',
  `filename` varchar(200) NOT NULL default '',
  PRIMARY KEY  (`push_resource_svn_file_id`),
  KEY `push_resource_svn_id` (`push_resource_svn_id`),
  KEY `filename` (`filename`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;

--
-- Dumping data for table `Push_Resource_SVN_File`
--


-- --------------------------------------------------------

--
-- Table structure for table `Push_Resource_Script`
--

DROP TABLE IF EXISTS `Push_Resource_Script`;
CREATE TABLE `Push_Resource_Script` (
  `push_resource_script_id` int(65) NOT NULL auto_increment,
  `push_request_id` int(65) NOT NULL default '0',
  `server_id` int(65) NOT NULL default '0',
```

```
  `command` text NOT NULL,
  `script_user` varchar(50) NOT NULL default '',
  `notes` text NOT NULL,
  `status` enum('waiting','pending','processing','pushed') NOT NULL default 'waiting',
  PRIMARY KEY  (`push_resource_script_id`),
  KEY `push_request_id` (`push_request_id`),
  KEY `server_id` (`server_id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;

--
-- Dumping data for table `Push_Resource_Script`
--


-- --------------------------------------------------------

--
-- Table structure for table `Push_Result`
--

DROP TABLE IF EXISTS `Push_Result`;
CREATE TABLE `Push_Result` (
  `push_result_id` int(65) NOT NULL auto_increment,
  `resource_type`
enum('Push_Branch','Push_Database','Push_Resource_Database','Push_Resource_Cron','Push_Re
source_SVN','Push_Resource_Script','Server') NOT NULL default 'Push_Branch',
  `resource_id` int(65) NOT NULL default '0',
  `push_branch_id` int(65) default NULL,
  `date_pushed` timestamp NOT NULL default CURRENT_TIMESTAMP on update CURRENT_TIMESTAMP,
  `push_command` text,
  `push_result` text NOT NULL,
  PRIMARY KEY  (`push_result_id`),
  KEY `date_pushed` (`date_pushed`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;

--
-- Dumping data for table `Push_Result`
--


-- --------------------------------------------------------

--
-- Table structure for table `Push_User`
--

DROP TABLE IF EXISTS `Push_User`;
CREATE TABLE `Push_User` (
  `push_user_id` int(65) NOT NULL auto_increment,
  `username` varchar(100) NOT NULL default '',
  `password` varchar(255) NOT NULL default '',
  `first_name` varchar(100) NOT NULL default '',
  `last_name` varchar(100) NOT NULL default '',
  `email` varchar(150) NOT NULL default '',
  `phone` varchar(10) default NULL,
  `send_messages` tinyint(4) NOT NULL default '1',
  `requires_admin_approval` tinyint(4) NOT NULL default '0',
  `disabled` tinyint(4) NOT NULL default '0',
  `svn_username` varchar(75) NOT NULL default '',
  PRIMARY KEY  (`push_user_id`),
  KEY `username` (`username`),
  KEY `password` (`password`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=10 ;

--
-- Dumping data for table `Push_User`
--
```

```
INSERT INTO `Push_User` VALUES (9, 'bott', 'I÷KŽ?µŒÂµí¬†‹', 'Bryce', 'Ott',
'bott@doba.com', '8013803641', 1, 0, 0, '');
INSERT INTO `Push_User` VALUES (1, 'guest', '', 'guest', 'user', '', '', 0, 1, 1, '');
INSERT INTO `Push_User` VALUES (2, 'deployer', '', 'deployer', 'user', '', '', 0, 1, 1,
'');

-- --------------------------------------------------------

--
-- Table structure for table `Push_User_Group`
--

DROP TABLE IF EXISTS `Push_User_Group`;
CREATE TABLE `Push_User_Group` (
  `push_user_id` int(65) NOT NULL default '0',
  `user_group_id` int(65) NOT NULL default '0',
  PRIMARY KEY  (`push_user_id`,`user_group_id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--
-- Dumping data for table `Push_User_Group`
--

INSERT INTO `Push_User_Group` VALUES (2, 1);
INSERT INTO `Push_User_Group` VALUES (2, 2);
INSERT INTO `Push_User_Group` VALUES (3, 1);
INSERT INTO `Push_User_Group` VALUES (3, 2);
INSERT INTO `Push_User_Group` VALUES (4, 1);
INSERT INTO `Push_User_Group` VALUES (4, 2);
INSERT INTO `Push_User_Group` VALUES (7, 1);
INSERT INTO `Push_User_Group` VALUES (7, 2);
INSERT INTO `Push_User_Group` VALUES (8, 1);
INSERT INTO `Push_User_Group` VALUES (8, 2);
INSERT INTO `Push_User_Group` VALUES (9, 1);
INSERT INTO `Push_User_Group` VALUES (9, 2);

-- --------------------------------------------------------

--
-- Table structure for table `SVN_Project`
--

DROP TABLE IF EXISTS `SVN_Project`;
CREATE TABLE `SVN_Project` (
  `svn_project_id` int(65) NOT NULL auto_increment,
  `server_id` int(65) NOT NULL default '0',
  `project_name` varchar(255) NOT NULL default '',
  `root_url` varchar(255) NOT NULL default '',
  `branch_url` varchar(255) NOT NULL default '',
  `description` text NOT NULL,
  `admin_username` varchar(65) NOT NULL default '',
  `admin_password` varchar(65) NOT NULL default '',
  PRIMARY KEY  (`svn_project_id`),
  KEY `server_id` (`server_id`),
  KEY `project_name` (`project_name`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;

--
-- Dumping data for table `SVN_Project`
--


-- --------------------------------------------------------

--
-- Table structure for table `Server`
--

DROP TABLE IF EXISTS `Server`;
```

```
CREATE TABLE `Server` (
  `server_id` int(65) NOT NULL auto_increment,
  `hostname` varchar(255) NOT NULL default '',
  `server_type_id` int(65) NOT NULL default '0',
  `description` text NOT NULL,
  `allow_script_execution` tinyint(1) NOT NULL default '0',
  PRIMARY KEY  (`server_id`),
  KEY `hostname` (`hostname`),
  KEY `allow_script_execution` (`allow_script_execution`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;

--
-- Dumping data for table `Server`
--


-- --------------------------------------------------------

--
-- Table structure for table `Server_Group`
--

DROP TABLE IF EXISTS `Server_Group`;
CREATE TABLE `Server_Group` (
  `server_group_id` int(65) NOT NULL auto_increment,
  `server_group_name` varchar(255) NOT NULL default '',
  `next_server_group_id` int(65) NOT NULL default '0',
  `default_request_permission` tinyint(4) NOT NULL default '1',
  `default_view_permission` tinyint(4) NOT NULL default '1',
  `default_push_permission` tinyint(4) NOT NULL default '1',
  `description` text NOT NULL,
  PRIMARY KEY  (`server_group_id`),
  KEY `server_group_name` (`server_group_name`),
  KEY `next_server_group_id` (`next_server_group_id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;

--
-- Dumping data for table `Server_Group`
--


-- --------------------------------------------------------

--
-- Table structure for table `Server_Group_Cron_File`
--

DROP TABLE IF EXISTS `Server_Group_Cron_File`;
CREATE TABLE `Server_Group_Cron_File` (
  `server_group_id` int(65) NOT NULL default '0',
  `cron_file_id` int(65) NOT NULL default '0',
  PRIMARY KEY  (`server_group_id`,`cron_file_id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--
-- Dumping data for table `Server_Group_Cron_File`
--


-- --------------------------------------------------------

--
-- Table structure for table `Server_Group_Destination`
--

DROP TABLE IF EXISTS `Server_Group_Destination`;
CREATE TABLE `Server_Group_Destination` (
  `server_group_destination_id` int(65) NOT NULL auto_increment,
  `server_group_id` int(65) NOT NULL default '0',
```

```
  `destination_type` enum('SVN_Project','Push_Database','Cron_File') NOT NULL default
'SVN_Project',
  `destination_id` int(65) NOT NULL default '0',
  PRIMARY KEY  (`server_group_destination_id`),
  KEY `server_group_id` (`server_group_id`),
  KEY `destination_type` (`destination_type`),
  KEY `destination_id` (`destination_id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;

--
-- Dumping data for table `Server_Group_Destination`
--


-- --------------------------------------------------------

--
-- Table structure for table `Server_Group_Script_Server`
--

DROP TABLE IF EXISTS `Server_Group_Script_Server`;
CREATE TABLE `Server_Group_Script_Server` (
  `server_group_id` int(65) NOT NULL default '0',
  `server_id` int(65) NOT NULL default '0',
  PRIMARY KEY  (`server_group_id`,`server_id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--
-- Dumping data for table `Server_Group_Script_Server`
--


-- --------------------------------------------------------

--
-- Table structure for table `Server_Type`
--

DROP TABLE IF EXISTS `Server_Type`;
CREATE TABLE `Server_Type` (
  `server_type_id` int(65) NOT NULL auto_increment,
  `server_type_name` varchar(255) NOT NULL default '',
  `is_default` tinyint(4) NOT NULL default '0',
  `requires_admin_approval` tinyint(4) NOT NULL default '0',
  `description` text NOT NULL,
  PRIMARY KEY  (`server_type_id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=4 ;

--
-- Dumping data for table `Server_Type`
--

INSERT INTO `Server_Type` VALUES (1, 'SVN', 1, 1, 'The server type corresponds to an SVN
repository or local copy of a repository location.');
INSERT INTO `Server_Type` VALUES (2, 'Database', 1, 1, 'This is a database server which
houses a MySQL database.');
INSERT INTO `Server_Type` VALUES (3, 'Cron', 1, 1, 'This server hosts a cron service for
automating scripting tasks.');

-- --------------------------------------------------------

--
-- Table structure for table `User_Group`
--

DROP TABLE IF EXISTS `User_Group`;
CREATE TABLE `User_Group` (
  `user_group_id` int(65) NOT NULL auto_increment,
  `group_name` varchar(255) NOT NULL default '',
```

```
  `is_default_group` tinyint(4) NOT NULL default '0',
  `default_request_permission` tinyint(4) NOT NULL default '1',
  `default_view_permission` tinyint(4) NOT NULL default '1',
  `default_push_permission` tinyint(4) NOT NULL default '1',
  `description` text NOT NULL,
  PRIMARY KEY  (`user_group_id`),
  KEY `group_name` (`group_name`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=3 ;

--
-- Dumping data for table `User_Group`
--

INSERT INTO `User_Group` VALUES (1, 'Administrators', 0, 1, 1, 1, 'This group is for site
administrators who have full control over changing settings and processing push
requests.');
INSERT INTO `User_Group` VALUES (2, 'Engineers', 1, 1, 1, 0, 'This is a group for all the
engineering types. By default they can create and view requests.');

-- --------------------------------------------------------

--
-- Table structure for table `User_Group_Server_Group`
--

DROP TABLE IF EXISTS `User_Group_Server_Group`;
CREATE TABLE `User_Group_Server_Group` (
  `user_group_id` int(65) NOT NULL default '0',
  `server_group_id` int(65) NOT NULL default '0',
  PRIMARY KEY  (`user_group_id`,`server_group_id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--
-- Dumping data for table `User_Group_Server_Group`
--


-- --------------------------------------------------------

--
-- Table structure for table `User_Group_Source`
--

DROP TABLE IF EXISTS `User_Group_Source`;
CREATE TABLE `User_Group_Source` (
  `user_group_source_id` int(65) NOT NULL auto_increment,
  `user_group_id` int(65) NOT NULL default '0',
  `svn_project_id` int(65) NOT NULL default '0',
  PRIMARY KEY  (`user_group_source_id`),
  KEY `user_group_id` (`user_group_id`),
  KEY `svn_project_id` (`svn_project_id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=1 ;

--
-- Dumping data for table `User_Group_Source`
--
```

Appendix B: System Objects

The following is a list of objects used by the deployment tool system. The majority of them are business objects with their matching persister that directly correspond to tables in the database. There are also addition business objects that function to perform other essential business logic in the system.

B.1 AutoPushBranchCreation

This object is used for managing settings and functionality dealing with scheduling for the automatic creation of testing branches. It stores the dates and times when automatic push branches should be created.

B.2 AutoPushBranchCreationPersister

This object handles the persistence and retrieval of AutoPushBranchCreation objects.

B.3 AutoPushBranch

This object is used for managing the settings dealing with the automatic creation of testing branches. It defines from which SVN source the testing branch should be created and where it should eventually be pushed.

B.3 AutoPushBranchPersister

This object handles the persistence and retrieval of AutoPushBranch objects.

B.4 BaseObject

All other objects in the tool inherit from this class. It provides core functionality such as dirty flags on all object attributes, and the use of internal errors and warnings.

B.5 CronFile

This object is used to store and manage functionality dealing with cron file resources that are managed in the deployment tool. It defines where the cron file exists and provides functions for performing actions on it.

B.6 CronFilePersister

This object handles the persistence and retrieval of CronFile objects.

B.7 DatabaseConnection

This object extends the core *Database* object (see below) and is used to connect to and manage tasks associated with database resources being managed by the deployment tool. It is meant to separate database functionality dealing with these managed resources from that employed by the tool itself in the fulfillment of its functionalities. Although the tool currently only supports MySQL database resources, this object is utilized to

allow for the future addition of connection functionality for other database platforms.

B.8 Database

This is the core database object that is used for connecting to, executing queries on, and retrieving results from a MySQL database. Its functionality is utilized to interact with the tool's database as well as those of managed resources.

B.9 DatabaseSchema

The purpose of this object is to provide functionality for handling the management of database schemas that can be used when creating the database for new testing or final push branches. It allows the utilization of uploaded schema text files containing SQL or the user of schema files that are versioned in SVN.

B.10 Database Schema Persister

This object handles the persistence and retrieval of DatabaseSchema objects.

B.11 DBPersister

This is the core object for all persister objects that utilize database persistence. It provides core functions and attributes that are uniform for all database persistence such as result sets, a database object, the number of

affected rows, query execution, etc. This object extends the Persister base class (see below).

B.12 LoginLock

This object is used as part of the login lockout system designed to prevent brute force login attacks on the system. When the specified threshold for failed logins with a particular username has been reached, an instance of this object will be created for the username and will remain active until its expiration timeout has been reached.

B.13 LoginLockPersister

This object handles the persistence and retrieval of LoginLock objects.

B.14 LoginLog

The primary function of this object is to manage functionality associated with the accounting of user logins and login attempts. Each time a valid username is user to attempt login to the site, it is recorded in the database. This object plays a core role in determining when a LoginLock object should be created for a username with too many failed login attempts.

B.15 LoginLogPersister

This object handles the persistence and retrieval of LoginLog objects.

B.16 Message

The storage and functionality associated with system messages is the main purpose of this object. These messages are those sent to push users when certain events occur (such as the processing of a push request), or those that they can send to each other.

B.17 MessagePersister

This object handles the persistence and retrieval of Message objects.

B.18 MessageRecipient

A MessageRecipient object is used to manage the users that are recipients of a particular message and is employed to prevent the same message sent to different users from being replicated unnecessarily.

B.19 MessageRecipientPersister

This object handles the persistence and retrieval of MessageRecipient objects.

B.20 Persister

This is the core class object inherited by all persisters in the system. It provides an interface for the functions that should be employed by all persisters.

B.21 ProgramSetting

Program settings are those settings global to the entire program and are persisted in a generic fashion in the deployment database. This object provides access to those program settings and their values.

B.22 ProgramSettingPersister

This object handles the persistence and retrieval of ProgramSetting objects.

B.23 PushBranch

Since the various versions of managed resources are stored in SVN branches, this object is critical to the system. Push branches can be either testing branches or their final destinations. This object interacts with SVN to provide access to manipulate those push branches and their settings.

B.24 PushBranchPersister

This object handles the persistence and retrieval of PushBranch objects.

B.25 PushDatabase

This object defines a database resource being managed by the deployment tool, including the server it is hosted on and credentials to authenticate to it. It provides functions that allow the credentials to be validated and a connection to the database to be retrieved for running queries against it.

B.26 PushDatabasePersister

This object handles the persistence and retrieval of PushDatabase objects.

B.27 PushRequest

This object is involved in managing all push requests submitted to the system by push users. It provides access to the contents of a push request including any of the resources it contains for push. In addition, it can access any results that may have been received due to the push being processed.

B.28 PushRequestPersister

This object handles the persistence and retrieval of PushRequest objects.

B.29 PushRequestPushBranch

This object ties a push request to a specific destination push branch where it will be pushed to. Push requests can be configured to push to multiple destination push branches.

B.30 PushRequestPushBranchPersister

This object handles the persistence and retrieval of PushRequestPushBranch objects.

B.31 PushResourceCron

As part of a push request, a cron file is one of the possible resources that can be pushed. A cron file change that has been added to a push request will be stored and managed by this object.

B.32 PushResourceCronPersister

This object handles the persistence and retrieval of PushResourceCron objects.

B.33 PushResourceDatabase

Database changes are one of the resources that can be pushed through the deployment tool. When database SQL changes are added to a push request, this is the object that stores and manages them.

B.34 PushResourceDatabasePersister

This object handles the persistence and retrieval of PushResourceDatabase objects.

B.35 PushResourceScript

As part of a push request, scripts can be run on a allowed target host. These script commands are added to the push request and managed via this object.

B.36 PushResourceScriptPersister

This object handles the persistence and retrieval of PushResourceScript objects.

B.37 PushResourceSvnFile

The final and most common resources managed by the deployment tool are SVN files. These are pushed via the filename and revision number. This object works in conjunction with the PushResourceSvn object to define the files from a particular revision that should be pushed.

B.38 PushResourceSvnFilePersister

This object handles the persistence and retrieval of PushResourceSvnFile objects.

B.39 PushResourceSvn

As part of a push request, SVN resources can be deployed to a destination branch, so this object is used to define which revision number from which SVN source the resources are found in. It works closely with the PushResourceSvnFile object.

B.40 PushResourceSvnPersister

This object handles the persistence and retrieval of PushResourceSvn objects.

B.41 PushResult

Once a push request or testing branch has been pushed to either its destination testing branch, for push requests, or final destination branch for push branches, the results of the push are stored and managed by this object. These results will include any errors. Each type of resource (SVN files, cron file, database changes, script, or push branch merge) will have its own instance of this object.

B.42 PushResultPersister

This object handles the persistence and retrieval of PushResult objects.

B.43 Push Server

This object is used to manage and access the various servers that are managed through the deployment tool. These can be servers hosting source SVN resources, or they could be hosting any of the possible resources that can be deployed (SVN files, databases, scripts, or cron files). A single PushServer object may be tied to several of these resources.

B.44 PushServerPersister

This object handles the persistence and retrieval of PushServer objects.

B.45 PushServerType

This object is utilized to classify what a PushServer object's primary type is. New server types can be added by a tool administrator to expand the functionality of the tool.

B.46 PushServerTypePersister

This object handles the persistence and retrieval of PushServerType objects.

B.47 PushUserGroup

In order to create affective permissions on resources in the system, push users are placed into groups that can then be given access to various system resources. This object manages the functionality of those groups.

B.48 PushUserGroupPersister

This object handles the persistence and retrieval of PushUserGroup objects.

B.49 PushUser

System users, whether they are normal push users or push administrators, are managed through this object. It stores information about the user including their default permission settings.

B.50 PushUserPersister

This object handles the persistence and retrieval of PushUser objects.

B.51 RepositoryConnect

The purpose of this object is to create a generic interface for versioning objects. Currently the system only has the ability to manage Subversion resources, but this object provides a versioning system platform independent method of calling those functionalities so that in the future support can more easily be added for other versioning systems. This object basically just forwards on the function calls to the object specified by the configured versioning platform.

B.52 ServerGroupCronFile

This object is used to define which cron file resources that a particular server group has access to, so that users that are members of that group can utilize functionality pertaining to those cron files.

B.53 ServerGroupCronFilePersister

This object handles the persistence and retrieval of ServerGroupCronFile objects.

B.54 ServerGroupDestination

This object manages the definition of SVN final destination branches that are affected by the settings on a particular server group. Thus a user that has access to the server group tied to this object will be able to push to testing branches with a final destination branch that matches this object.

B.55 ServerGroupDestinationPersister

This object handles the persistence and retrieval of ServerGroupDestination objects.

B.56 ServerGroup

In addition to users being put into groups to more effectively manage their permissions, servers and their corresponding resources can also be put into groups for the same purpose. This object defines a server group and the resources that it contains so that their permissions can be managed in a larger scope.

B.57 ServerGroupPersister

This object handles the persistence and retrieval of ServerGroup objects.

B.58 ServerGroupScriptServer

Servers that allow script execution that are defined as part of a server group are managed by this object. A push user that has access to a server group linked to this object can push script changes to that script server.

B.59 ServerGroupScriptServerPersister

This object handles the persistence and retrieval of ServerGroupScriptServer objects.

B.60 Session

This is the object that manages what is set in the PHP session. This mostly applies to pages that the user must login to access, and is used to persist settings across pages.

B.61 SvnConnect

This object is called by the RepositoryConnect object and is the specific implementation of its functionality for Subversion resources. It allows the system to fully interact with and manage resources versioned by Subversion, including any error and output handling.

B.62 SvnProject

This object is used to manage SVN projects that have been added to the deployment tool to be used as either SVN sources or destinations for a push request or push branch.

B.63 SvnProjectPersister

This object handles the persistence and retrieval of SvnProject objects.

B.64 UserGroup

In order to more effectively manage permissions that users have to the resources managed by the deployment tool, this object is used to access user groups that define those permissions. Users can be added to a user group to gain access to its resources.

B.65 UserGroupPersister

This object handles the persistence and retrieval of UserGroup objects.

B.66 UserGroupServerGroup

This object is used to tie user groups, which define a list of users with access to a list of resources, to server groups, which define the permissions on a list of resources.

B.67 UserGroupServerGroupPersister

This object handles the persistence and retrieval of UserGroupServerGroup objects.

B.68 UserGroupSource

This object defines which SVN source repositories that a particular user group has access to.

B.69 UserGroupSourcePersister

This object handles the persistence and retrieval of UserGroupSource objects.

B.70 Version_CSS

Used for marking the cache as dirty for CSS files when a new version of the file is promoted, forcing the client to download the newer version.

B.71 Version_Script

Used for marking the cache as dirty for JavaScript files when a new version of the file is promoted, forcing the client to download the newer version.

Appendix C: Destination SVN Server Scripts

C1. processsvnupdates.php

```
<?php
/**
 * This script is used by a destination server to process svn updates that should be run
on it. It should be cronned to run frequently,
 *      and must be able to connect to the database storing the deployment tool
information. It should by cronned to run in conjunction with
 *      the script cron, otherwise resource discrepencies may result.
 *
 * This script is used to be able to store the results of SVN updates on the destination
server into the deployment database, and will
 *      create a record in the Push_Result table only when the SVN update returns
results.
 *
 * In order to run properly as different users, this script must be run as a user with
sudo access.
 * TODO:  Find a safer way to do this.
 */

require_once('svnsettings.php');



/**
 * This function will handle an error by checking if an email should be sent with the
error information and then dying gracefully.
 *
 * @param string $errormsg The error message to email and display.
 * @param string $makedie If true, the script will die with the error.
```

```
 */
function handleError($errormsg, $makedie=true)  {
        //if the ERROR_EMAIL define is set, send an email
        if (ERROR_EMAIL != '') {
                $errormsg = "The following error occurred on
'".$_SERVER['SERVER_NAME']."' at ".date('Y-m-d H:i:s')."\n".$errormsg;
                mail(ERROR_EMAIL, 'DEPLOYMENT TOOL ERROR Running processsvnupdates.php ON
'.$_SERVER['SERVER_NAME'], $errormsg);
        }

        if($makedie)  {
                //die with the error message
                die($errormsg);
        }

        echo $errormsg;
}

//go through each working copy location and process SVN update
foreach ($workingcopies as $wc)  {
        //output an error if things have not been properly defined
        if(empty($wc['localpath']) || empty($wc['user']) || empty($wc['branchid']))  {
                handleError("Working copy not properly configured in svnsettings.php!\n".
                                        "    localpath:  ".$wc['localpath']."\n".
                                        "    user:  ".$wc['user']."\n".
                                        "    branchid:  ".$wc['branchid']."\n", false);
        }
        else  {
                //get the current user
                $results = array();
                exec('whoami', $results, $returnval);

                //if current user is the same, don't use sudo
                if($results[0] == $wc['user'])  {
                        $cmd = 'cd '.$wc['localpath'].'; svn update';
```

```
                }
                //otherwise use sudo
                else  {
                        $cmd = 'cd '.$wc['localpath'].'; sudo -u '.$wc['user'].' svn
update';
                }
                $results = array();
                exec($cmd, $results, $returnval);

                //check for error
                $msg = '';
                if($returnval !== 0)  {
                        $msg = "There was an error processing the SVN update. An error
code of '".$returnval."' was returned from the command.\n".
                                        "Command: ".$cmd."\n".
                                        "Output:  ".implode("\n", $results);
                }
                //if there was nothing updated, don't store the results
                else if(!empty($results))  {
                        //makse
                        $strresults = implode("\n", $results);
                        //if the results do not have 'At revision ' or they do and also
don't have 'Updated ' there is no need to report results since no
                        //changes occurred. This should report properly if externals
exist.
                        if((strpos(strtolower($strresults), 'at revision ') === false) ||
(strpos(strtolower($strresults), 'at revision ') !== false &&
                          strpos($strresults, 'Updated ') !== false))  {
                                $msg = $strresults;
                        }
                }

                //if there were results, store them to the database
                if(!empty($msg))  {
                        //establish the connection to the DB
```

```
                        $linkid = mysql_connect(DATABASE_HOSTNAME, DATABASE_USERNAME,
DATABASE_PASSWORD);
                        if(!$linkid)  {
                                handleError("Could not connect to '".DATABASE_NAME."' on
'".DATABASE_HOSTNAME."':  ".mysql_error()."\n");
                        }

                        //escape any SQL chars in the message
                        $msg = mysql_real_escape_string($msg);
                        $sql = "INSERT INTO ".DATABASE_NAME.".Push_Result (resource_type,
resource_id, push_branch_id, date_pushed, push_command, push_result) ".
                                        "VALUES('".PUSH_RESULT_RESOURCE_TYPE."',
'".PUSH_SERVER_ID."', '".$wc['branchid']."', '".date('Y-m-d H:i:s')."', '".
                                                $cmd."', '".$msg."')";

                        //execute the query
                        $res = mysql_query($sql);
                        //check for error
                        if($res === false)  {
                                handleError("Error inserting SVN updates results into
deployment DB using sql:  ".$sql."\nError:  ".mysql_error()."\n");
                        }
                }
        }
}

?>
```

C.2 svnsettings.php

```
<?php
/**
 * This file stores the settings needed by a destination svn server to connect to the
deployment database and properly store any results
```

```
 *      from updating.
 *
 * It should be edited on each destination svn server to make it specific to that
server's settings. Working copy locations that are added
 *      to the $workingcopies variable should have been checked out previously from their
respective branches with the authentication cached,
 *      so that issuing the 'svn update' command as the folder's owner will successfully
execute an update.
 */

//do not edit this value
define('PUSH_RESULT_RESOURCE_TYPE', 'Server');

//The server_id of this destination server as it appears in the Server table of the
deployment DB
define('PUSH_SERVER_ID', 0);

//The database where the deployment settings are stored. The database user needs INSERT
access to Push_Result from the host where this script
//      is running (see the database/deploygrant.sql file).
define('DATABASE_HOSTNAME', 'db.hostname');  //edit this to match the server hosting the
deployment DB
define('DATABASE_USERNAME', 'svnserver');  //do not change this username
define('DATABASE_PASSWORD', 'mypassword');  //change this to the match the password given
to the 'svnserver' user
define('DATABASE_NAME', 'deploytool');  //do not change this database name

//Where to send error emails if they occur
define('ERROR_EMAIL', '');  //add an email address to send errors in the script

//the following are the locations of working copies that need to be SVN updated on the
server and their corresponding push branch id's from
//      the deployment table.
$workingcopies = array(
        //for multiple working copies on the server, duplicate the following array
```

```
        array(
                'localpath' => '',  //the local path where the working copy is located
(make sure its literal)
                'user' => '',  //the user who the svn location was checked out as
                'branchid' => '',  //the branchid that the working copy was checked out
from as it appears in the Push_Branch table of the deployment database
        ),
);
?>
```

Appendix D: Destination Cron Server Scripts

D.1 processcronchanges.php

```
<?php
/**
 * This script is used by a destination server to process changes to its crons. It should
be cronned to run frequently, and must be able
 *      to connect to the database storing the deployment tool information.
 *
 * In order to write to the location where Cron files are stored, this script must be run
as root. In addition, the cron files must be
 *      orginally checked out from SVN so that the default parameters can be used to
update them.
 */

require_once('cronsettings.php');



/**
 * This function will handle an error by checking if an email should be sent with the
error information and then dying gracefully.
 *
 * @param string $errormsg The error message to email and display.
 */
function handleError($errormsg, $makedie=true)  {
        //if the ERROR_EMAIL define is set, send an email
        if (ERROR_EMAIL != '') {
                $errormsg = "The following error occurred on
'".$_SERVER['SERVER_NAME']."' at ".date('Y-m-d H:i:s')."\n".$errormsg;
```

```
                mail(ERROR_EMAIL, 'DEPLOYMENT TOOL ERROR Running processcronchanges.php
ON '.$_SERVER['SERVER_NAME'], $errormsg);
        }
        //die with the error message
        if($makedie)  {
                die($errormsg);
        }

        echo $errormsg;
}

//SQL to check if there are any scripts to run
$sql = "SELECT push_resource_cron_id, revision, server_filepath, cron_user FROM
".DATABASE_NAME.".Push_Resource_Cron AS prc INNER JOIN ".
                DATABASE_NAME.".Cron_File AS cf ON prc.cron_file_id=cf.cron_file_id WHERE
cf.server_id='".PUSH_SERVER_ID.
                "' AND prc.status ='pending'";

//establish the connection to the DB
$linkid = mysql_connect(DATABASE_HOSTNAME, DATABASE_USERNAME, DATABASE_PASSWORD);
if(!$linkid)  {
        handleError("Could not connect to '".DATABASE_NAME."' on '".DATABASE_HOSTNAME."':
".mysql_error()."\n");
}
$result = mysql_query($sql);
//check for error
if($result === false)  {
        handleError("Error checking for pending crons using sql:  ".$sql."\nError:
".mysql_error()."\n");
}

//go through each pending cron
while(list($prcronid, $revision, $serverfilepath, $cronuser) =
mysql_fetch_array($result))  {
        //set the resource status to 'processing'
```

```
        $sql = "UPDATE ".DATABASE_NAME.".Push_Resource_Cron SET status='processing' WHERE
push_resource_cron_id='".$prcronid."'";
        if(mysql_query($sql) === false)  {
                handleError("Could not set status of push_resource_cron_id
'".$prcronid."' to 'processing' using sql:  ".$sql."\nError:  ".mysql_error()."\n");
        }

        //get the current user
        $userres = array();
        exec('whoami', $userres, $returnval);

        //update the cron from SVN as the specified user
        $results = array();
        if($cronuser != 'root' && $userres[0] != $cronuser)  {
                $fullcommand = 'sudo -u '.$cronuser.' ';
        }
        //escape the params
        $fullcommand .= 'svn update -r'.escapeshellarg($revision).'
'.escapeshellarg($serverfilepath);
        //execute the command
        $results = array();
        exec($fullcommand, $results);
        $results = implode("\n", $results);

        //store the result into the database
        $sql = "INSERT INTO ".DATABASE_NAME.".Push_Result(resource_type, resource_id,
push_branch_id, date_pushed, push_command, push_result) ".
                        "VALUES('Push_Resource_Cron', '".$prcronid."', null, '".date('Y-
m-d H:i:s')."', '".mysql_real_escape_string($fullcommand,
                                $linkid)."', '".mysql_real_escape_string($results,
$linkid)."')";
        if(mysql_query($sql) === false)  {
                handleError("Could not set PushResult of push_resource_cron_id
'".$prcronid."' using sql:  ".$sql."\nError:  ".mysql_error()."\n");
        }
```

```
        //set the resource status to 'pushed'
        $sql = "UPDATE ".DATABASE_NAME.".Push_Resource_Cron SET status='pushed' WHERE
push_resource_cron_id='".$prcronid."'";
        if(mysql_query($sql) === false)  {
                handleError("Could not set status of push_resource_cron_id
'".$prcronid."' to 'pushed' using sql:  ".$sql."\nError:  ".mysql_error()."\n");
        }
}

?>
```

D.2 cronsettings.php

```
<?php
/**
 * This file stores the settings needed by a destination cron server to connect to the
deployment database and properly process its
 *      requests.
 *
 * It should be edited on each destination cron server to make it specific to that
server's settings.
 */

//The server_id of this destination server as it appears in the Server table of the
deployment DB
define('PUSH_SERVER_ID', 0);

//The database where the deployment settings are stored. The 'cronserver' user needs
SELECT access to the Push_Resource_Cron and
//      Cron_File tables, UPDATE access to Push_Resource_Cron, and INSERT access to
Push_Results (see the database/deploygrant.sql file)
define('DATABASE_HOSTNAME', 'db.hostname');  //edit this to match the server hosting the
deployment DB
```

```
define('DATABASE_USERNAME', 'cronserver');  //do not change this username
define('DATABASE_PASSWORD', 'mypassword');  //change this to the match the password given
to the 'cronserver' user
define('DATABASE_NAME', 'deploytool');  //do not change this database name

//Where to send error emails if they occur
define('ERROR_EMAIL', '');  //add an email address to send errors in the script

?>
```

Appendix E: Destination Script Server Scripts

E.1 processscripts.php

```
<?php
/**
 * This script is used by a destination server to process scripts that should be run on
it. It should be cronned to run frequently, and
 *      must be able to connect to the database storing the deployment tool information.
It shoulded by cronned to run in conjunction with
 *      the SVN update cron, otherwise resource discrepencies may result.
 *
 * TODO:  Investigate a more secure way to do this.
 * In order for the users specified in Push Requests to be able to have commands executed
as them, this script must be run by a user
 *      with access to sudo, or as root.
 */

require_once('scriptsettings.php');



/**
 * This function will handle an error by checking if an email should be sent with the
error information and then dying gracefully.
 *
 * @param string $errormsg The error message to email and display.
 */
function handleError($errormsg, $makedie=true)  {
        //if the ERROR_EMAIL define is set, send an email
        if (ERROR_EMAIL != '') {
```

```
                $errormsg = "The following error occurred on
'".$_SERVER['SERVER_NAME']."' at ".date('Y-m-d H:i:s')."\n".$errormsg;
                mail(ERROR_EMAIL, 'DEPLOYMENT TOOL ERROR Running processscripts.php ON
'.$_SERVER['SERVER_NAME'], $errormsg);
        }
        //die with the error message
        if($makedie)  {
                die($errormsg);
        }

        echo $errormsg;
}

//SQL to check if there are any scripts to run
$sql = "SELECT push_resource_script_id, command, script_user FROM
".DATABASE_NAME.".Push_Resource_Script WHERE server_id='".
                        PUSH_SERVER_ID."' AND status ='pending'";

//establish the connection to the DB
$linkid = mysql_connect(DATABASE_HOSTNAME, DATABASE_USERNAME, DATABASE_PASSWORD);
if(!$linkid)  {
        handleError("Could not connect to '".DATABASE_NAME."' on '".DATABASE_HOSTNAME."':
".mysql_error()."\n");
}
$result = mysql_query($sql);
//check for error
if($result === false)  {
        handleError("Error checking for pending scripts using sql:  ".$sql."\nError:
".mysql_error()."\n");
}

//go through each pending script
while(list($prscriptid, $command, $scriptuser) = mysql_fetch_array($result))  {
        //set the resource status to 'processing'
```

```
        $sql = "UPDATE ".DATABASE_NAME.".Push_Resource_Script SET status='processing'
WHERE push_resource_script_id='".$prscriptid."'";
        if(mysql_query($sql) === false)  {
                handleError("Could not set status of push_resource_script_id
'".$prscriptid."' to 'processing' using sql:  ".$sql."\nError:  ".mysql_error()."\n");
        }

        //get the current user
        $userres = array();
        exec('whoami', $userres, $returnval);

        //run the command as the specified user
        $results = array();
        if($scriptuser != 'root' && $userres[0] != $scriptuser)  {
                $fullcommand = 'sudo -u '.$scriptuser.' ';
        }
        //these args are not escaped because of the danger of causing undesireable
effects in the command
        //as such, COMMANDS SHOULD BE CAREFULLY REVIEWED BEFORE BEING APPROVED FOR PUSH
        $fullcommand .= $command;
        $results = array();
        exec($fullcommand, $results);
        $results = implode("\n", $results);

        //store the result into the database
        $sql = "INSERT INTO ".DATABASE_NAME.".Push_Result(resource_type, resource_id,
push_branch_id, date_pushed, push_command, push_result) ".
                        "VALUES('Push_Resource_Script', '".$prscriptid."', null,
'".date('Y-m-d H:i:s')."', '".mysql_real_escape_string($fullcommand,
                                $linkid)."', '".mysql_real_escape_string($results,
$linkid)."')";
        if(mysql_query($sql) === false)  {
                handleError("Could not set PushResult of push_resource_script_id
'".$prscriptid."' using sql:  ".$sql."\nError:  ".mysql_error()."\n");
        }
```

```
        //set the resource status to 'pushed'
        $sql = "UPDATE ".DATABASE_NAME.".Push_Resource_Script SET status='pushed' WHERE
push_resource_script_id='".$prscriptid."'";
        if(mysql_query($sql) === false)  {
                handleError("Could not set status of push_resource_script_id
'".$prscriptid."' to 'pushed' using sql:  ".$sql."\nError:  ".mysql_error()."\n");
        }
}

?>
```

E.2 scriptsettings.php

```
<?php
/**
 * This file stores the settings needed by a destination script server to connect to the
deployment database and properly process its
 *      requests.
 *
 * It should be edited on each destination script server to make it specific to that
server's settings.
 */

//The server_id of this destination server as it appears in the Server table of the
deployment DB
define('PUSH_SERVER_ID', 0);

//The database where the deployment settings are stored. The database user needs SELECT
access to the Push_Resource_Script
//      table, UPDATE access to Push_Resource_Script, and INSERT access to Push_Results
(see the database/deploygrant.sql file)
define('DATABASE_HOSTNAME', 'db.hostname');  //edit this to match the server hosting the
deployment DB
```

```
define('DATABASE_USERNAME', 'scriptserver');  //do not change this username
define('DATABASE_PASSWORD', 'mypassword');  //change this to the match the password given
to the 'scriptserver' user
define('DATABASE_NAME', 'deploytool');  //do not change this database name

//Where to send error emails if they occur
define('ERROR_EMAIL', '');  //add an email address to send errors in the script

?>
```

Appendix F: Industry Survey Questions

1. Please give your name, title, and the company you work for.

2. Does your company use one or more dynamic scripting languages to serve web content (may be internal or external)? If so, which ones? About how many servers are each of the languages server from?

3. Does your company use Crons or other scheduled services to manage resources? Please describe what you use.

4. Does your company use versioning software to manage source code and/or configuration files, etc.? If so, which ones?

5. What types of software/application resources does your company need to deploy (ie. web code, databases, etc.)?

6. Of the resources stated in #2, which of them are accessed directly or indirectly from the web?

7. What are the sizes of the systems that house the resources mentioned above (number of servers, DB hosts, etc.)?

8. What platforms are the resources mentioned above running on (OSes, DB'e, etc.)?

9. Please describe how your company deploys each of the resources mentioned above (ie. engineer manually uploads to each server, rpm, etc.).

10. Does your company have a Quality Assurance department? What role do they play in the approval process for when resources are ready to be deployed?

11. What role does management play in the approval process for when resources are ready to be deployed?

12. Is there anyone else who has a say in when resources are ready to be deployed?

13. How much interaction does the engineer or engineers who deploys resources have with software developers? How much interaction do they have with QA? How much interaction with management in regards to when resources are ready to be deployed?

14. Are there any shortcomings or limitations with the methodology or process that your company uses to deploy resources? If so, what are they?

15. Are there any improvements you would like to see in how your company deploys resources? If so, What are they?