ISSN (Online) : 2278-1021
ISSN (Print) : 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 6, June 2014

Efficient Automated Build and Deployment Framework with Parallel Process

Prachee Kamboj[1], Lincy Mathews[2]

Information Science and engineering Department, M. S. Ramaiah Institute of Technology, Bangalore, India [1, 2]

Abstract: Continuous integration has been in theory for a long time but successful practicing is still a dream. To resolve the integration problems and increase the customer satisfaction, it is important to follow the practice of continuous integration in software development. Doing manual build and deployment regularly results in lot of time consumption and thus becomes an inefficient process. This paper proposes architecture for build and deployment by following best practices. It aims to fully automate the build process by using some available tools for source management, binary management and compilation. Build architecture gives a method to integrate all tools in build server. Successful completion of build is followed by deployment. This paper also proposes a deployment architecture which is common for all development environments. Deployment architecture states three steps, required to do the deployment and also proposes optional features to enhance the deployment process.

Keywords: Automation, Build, Continuous Integration, Deployment, Parallel

I. INTRODUCTION

Software development is done in different environments. Generally industry follows four environments which are Development, Quality Assurance, Stage, and Production [1]. Before delivering the software to the customer it is essential to test the software and ensure the quality of software. The environments should not intersect as the activities between each group in software development should not interfere with one another. The development environment is used for developing the application. Users in development environment can expect the application to change often and be broken for extended periods of time. The Quality Assurance environment is used for testing. The staging environment is the dummy of production environment and is used to test the application for production environment without having users accessing the application. This environment is used to make sure security settings are correct and that other services on the user computer or server do not interfere with the application. The production environment is the environment where the users actually run the live application. Changes to this environment happen in periodic updates to the software after the application changes have passed through the previous three environments.
The other environments help ensure that the application is well tested and stable when it is released.
Build and deployment is an integral part of any software development process. Software build refers to the process of compiling the source code into a form which can be deployed in a computer or a server. In an environment like development and quality assurance, build and deployment can be done many times in a day also. It makes sense to make whole build and deployment process automated. Automation is being adopted by many organization for its unmatchable advantages compared to manual process. Fig 1 shows the block diagram of build and deployment automation on different development environments [1].

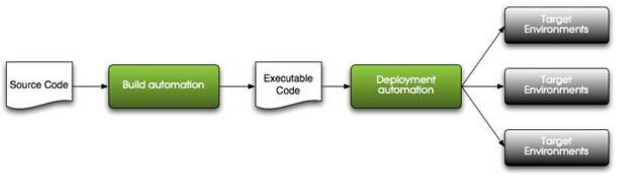


Figure 1: Build and Deployment automation for different development Environments

This paper proposes a new build and deployment framework suitable for enterprise. Framework has focused on automation and introduction of parallelism to increase efficiency and reduce response. Build and deployment process is same for all environments to avoid confusion for the users.

II. BUILD

A. Build Steps

Conventionally building software is referred to as converting the source code into executable file [2]. In this era of software development getting mature, it needs to do some more tasks which makes the build stable and more meaningful. It is an end-to-end process that comprises of many steps as shown in figure 2 below.

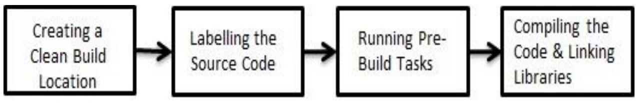

Creating a Clean Build Location
Labelling the Source Code
Running Pre-Build Tasks
Compiling the Code & Linking Libraries

Figure 2: Block diagram of Build Steps

Creating a Clean Build Location prepares an environment for the build process to execute. Same source code file can be modified by many users simultaneously. To maintain consistency, user's files are tagged. Original copies of all source files are labelled in baseline separately. Labelling the source code approves the source file which is consistent among all files. Running Pre-Build Tasks includes capturing metadata about the code like setting the version number for the output file, etc. Compiling the Code & Linking Libraries includes Unit and integration testing is also done before compilation to ensure reliability. Other types of testing can be done depending on the application requirement. At the end, it outputs the executable code which is ready for deployment.

B. Build Automation

Agile software development is a proven method adopted by many enterprises. Among the basic principle of agile software development methodology is contiguous integration [4] [5]. Continuous integration is a practice based on principle of detecting integration error in early stage. It is practiced by integrating developers codebase many times in a day. Usually integration is done at least on daily basis. Build is a repeated process, done in every environment before going live for the users of application. Even if one is not practicing continuous integration, build automation is required. Build automation helps in improving the software quality, reduces compile and linking the library time, eliminates redundant tasks [3], minimizes broken builds occurrence, reduces key personnel dependencies, improves traceability through logging the build steps etc.

C. Build Architecture

Figure 3 explains the build process. In build architecture steps 2 to 7 can be automated if one is practicing continuous Integration, else user can trigger the build manually to initiate the build so that steps 3 to 7 are automated.

The key component of this architecture is to set a build server. Build server is the next step to have a stable and robust build. [9] This server is the heart of the whole build process. Build server performs all major tasks like pulling down the latest labelled source code, acts as a continuous integration server by installing build tool which triggers build automatically, does the compilation of the source code etc. Build server will always have the clean code of the source code label. Build server will also publish the binary to binary repository management system. It even resolves the encoding dependency problem for binaries. Software development process produces mainly source code and binary artifacts (or executable code) which serve different purpose. Source configuration management system is commonly required where there are multiple developers working on the same project concurrently on same set of source file. It helps in source control, change management, and in version control. It keeps the track of the changes happening in the file, assigns a temporary number to the users file and keeps the original file preserved. When a developer is satisfied with the source file then source file is checked in to source configuration management server to create a new version of the file. If many users check in the file at same time then the source configuration management server combines the changes and is merged to create a new version of file.

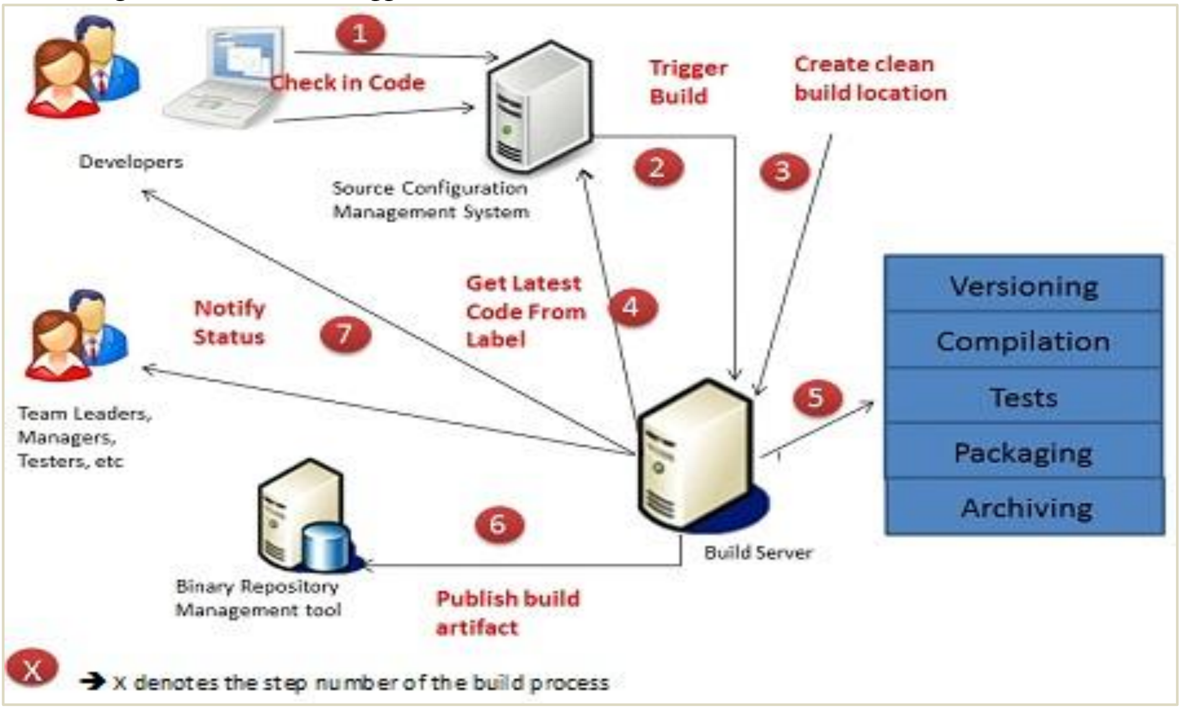

1
Check in Code
Developers
Source Configuration Management System
Trigger Build
2
Create clean build location
3
Get Latest Code From Label
4
Notify Status
7
Team Leaders, Managers, Testers, etc
5
Versioning
Compilation
Tests
Packaging
Archiving
Build Server
6
Binary Repository Management tool
Publish build artifact
X
X denotes the step number of the build process

Figure 3: Build Architecture

There are many ways to trigger build but as this paper talks about continuous integration practice, this step should be automated. Ideally as soon as developer commits the source code in source code repository, build should be triggered. Build can be scheduled for nightly basis and there should be scope of manually triggering the build. After this all steps should be automated and executed sequentially.

In build server, build tool creates a clean build location for the build and deletes old temporary build space that is not required for the current build. Build tool is configured to get the latest source code in the build server from source configuration management server. This source code is stored in build server and it should not be modified further in the build server. Compiling the code is the major step and the most time consuming step. Code is compiled base

on the compiler required by the application programming language. Unit testing can also be performed as part of this step. After successful completion of the testing, it should be archived with a meaningful label on it. It should have some numbering or a time stamp value to distinguish the file from the other builds.
Once the build is completed, archived file should be send to the Binary Repository Management server at a common location for all the builds of this application. Doing so helps other team members to download the file when required. If a build fails concerned person should be informed about the same. Build logs are also generated to find the reason of the build failure.

III. DEPLOYMENT

A. Deployment Steps

Software deployment refers to collection of activities which makes the software run on a particular or all environments to use. In general, software deployment process consists of interconnected tasks which are dependent. All software is unique in itself, it has different modules. Software applications are different and so is the different way to execute the module in software. It's hard to come up with a standard approach but a general approach can be proposed which then can be customized according to the requirements and characteristics of the project. Deployment has its list of general activities [11] as shown below in figure 4.

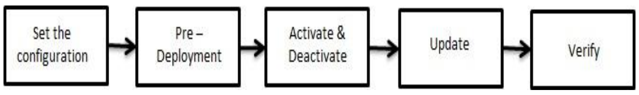


Figure 4: General Steps of Deployment

Before starting the deployment it's important to know the site and collect the information from where deployment activities actually have to be carried out. Sites have to be installed with the executables required by the application as part of pre-deployment step. For complex system, it is recommended to keep the installation ready as part of pre-deployment step. Executables need to be deactivated or stopped if started previously. This process replaces the earlier version of software system with the latest release. Verification of successful deployment is done at last as to follow the best practice.

B. Deployment Architecture

Build server is the same server used for build automation. This architecture describes four development environments; it can be extended for more environments if required. All servers are connected in private network but this architecture works fine for any networked devices. One prerequisite to this architecture is to have a common mounted point for all local servers of an environment. [12] Fulfilling this requirement reduce the execution time drastically, saves the server space and removes redundant copies of the file. To start the deployment, the user remotely logins to the build server and execute the main script that is the starting point of the deployment execution.

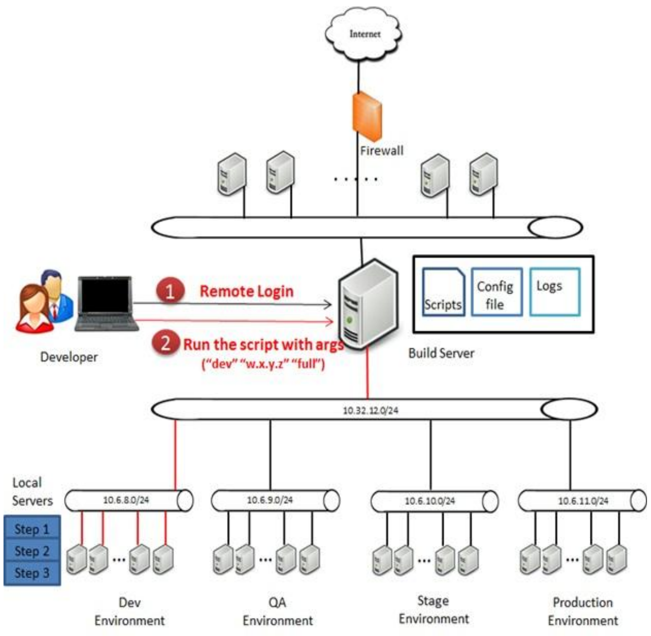


Figure 5: Deployment Architecture

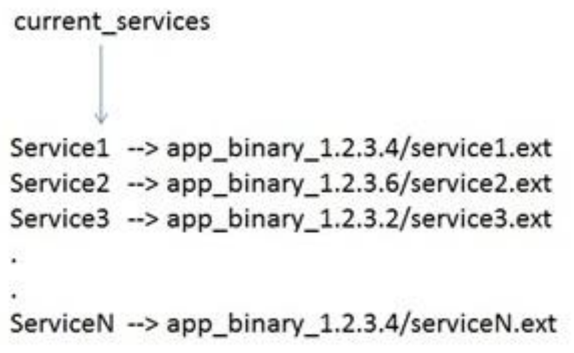
Script should take the required arguments like software development environment, version of the build and service name. Software development environment, value can be given as dev, qa, stage or production as shown in figure 5. Version of the build, is the part of label of the archived file stored in binary repository management server during the build process. Service name, value can be "full" or "name of the services". Giving "full" as the value of the service name, script does the full deployment. In full deployment, all the services of the software application are deployed. Giving "name of the services" as the value for the service name, script does partial deployment. In partial deployment, only services given in argument are deployed. So this situation arises to two cases deployment that is full deployment and partial deployment. Full deployment usually happen when all modules of software have gone for a big change, when there is a new environment introduced or there is change in configuration of the environment. In Partial deployment, happens only when some services have gone for a change and needs to be deployed again. Script also validates the correctness of the arguments from the configuration files like checking of the service name and etc. Once validating is done, script looks for the configuration file related to that software development environment to get the configuration details to execute step 1, step 2 and step 3 sequentially.

C. Step 1: Download and Extract

Download the binary from the binary repository management server based on the version number given in the argument that is w.x.y.z. Binary can be any form like tar, tar.gz , war, zip, etc. The type of file depends on packaging and achieving done in build process. Once the binary is downloaded to the desired location in the local server, then the file should be extracted in other folder. It is recommended to create folder name having version number on it. This step needs to be done only once in any server because server volume is mounted on a common mount point.

D. Step 2: Unmount and Mount

Once the extracted folder is available, it contains some files like jar, war, ear etc files, if project is in java and similarly for projects in other different languages. Let's refer these files as services of the project. This architecture introduces the concept of mounting the services of extracted folder to different folder. This is achieved by creating soft link. Let's call the folder as "current_services". In folder current_services create link pointing to the service file of the current build version number. This looks like as shown in figure 6.
In figure 6 first line stated as "Service1 --> app_binary_1.2.3.4/service1.ext" has its meaning as explained. Here **app_binary_1.2.3.4** is extracted folder from the binaries generated during the build. service1.ext is file in extracted folder. Service1 is the soft link created in current_services folder pointing to the app_binary_1.2.3.4/service1.ext


```
current_services

Service1  --> app_binary_1.2.3.4/service1.ext
Service2  --> app_binary_1.2.3.6/service2.ext
Service3  --> app_binary_1.2.3.2/service3.ext
.
.
ServiceN  --> app_binary_1.2.3.4/serviceN.ext
```

Figure 6: After Mount, for Partial Deployment

Deployment can be done without creating soft link. Let's discuss the requirement of creating soft link. It is advisable to deploy only the services whose source code has been changed compared with previous version, which is called partial deployment. Partial deployment decreases execution time. As shown in figure 6 the case of partial deployment, service pointers points to different version executables. In figure 6 each service has a soft link that is pointing to the currently running version executable in that environment. Unmount removes the current link and mount creates the link for the required version executable of that service. In case of full deployment all service pointers in current_services will be pointing to the same version executables. Even if a service is to be run in different servers then also this step is executed only once for a service because the server volume is mounted at common mount point.
Following this method gives the benefits of debugging basic information about deployment like to enquire the version running for a service. It's easy to check the current_services folder under which each service has a soft link pointing to a current running executable location. In figure 6, Service1 is running for build version 1.2.3.4, Service2 is running for build version 1.2.3.6 and similarly for other services also. In the example Service2 is running the latest code.

E. Step 3: Restart

A service can run on more than one server to resolve issues like load balancing etc. This step should be executed on all the servers where a service will run. Starting and stopping a service is application dependent. Every application has its own way of executing this step. The best practices recommended is to first stop the service followed by verification and only then start the service followed by verification. Verification should be done to get the confirmation of whether the service has stopped or started successfully and correctly in the correct server. To start a service, execute the service link created in step2 under current_services folder where link is actually pointing to the latest executables required.

F. Parallelism

In the proposed architecture of deployment, parallelism is introduced to increase efficiency and to reduce the execution time. Parallelism is introduced by creating process on a local server concurrently while reading the configuration file for that environment. Thus when process is created for local server1, then server sends the created process in background and server creates another process for server2. All process running in background are monitored simultaneously to obtain the logs.
As shown in figure 7 there is dependency in the step 1, step 2 and step 3 but internal steps can be executed concurrently. Once all background process executing internal steps of step 2 or step 3 have finished the execution, only then execute the next step.

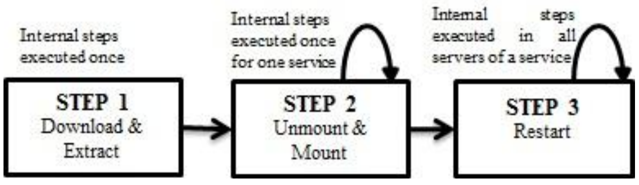

Internal steps executed once
Internal steps executed once for one service
Internal steps executed in all servers of a service
STEP 1 Download & Extract
STEP 2 Unmount & Mount
STEP 3 Restart

Figure 7: Frequency and Dependency in Steps of Deployment

G. Other Features

Maintaining logs is the most important part of automated deployment. If something goes wrong it's easy to track the occurrence of the fault. In the architecture proposed logs are maintained in the build server. It allows easy access to the logs as user has already remotely logged in build server. It's useful to give the report or deployment summary clearly mentioning the errors occurred as this can save users time. If a service is not deployed properly then it should be reverted back to the previous version. This makes the process more reliable. Deployment process can be extended to have access based on the role e.g. developers should be allowed to deploy only in the Development environment and not in Stage and Production environments. Sending emails at the ends of deployment to the concerned persons like manger, team leader etc. helps the authorities to track what is happening and who is doing it. If a new environment is introduced or configuration of an environment is changed then it shouldn't be a headache to incorporate the changes. A file is created in the remote server which keeps the track of who is deploying what and in which environment.

IV. BUILD AND DEPLOYMENT

In a company standard following Continuous Integration practice, build and deployment should be merged together. In build process, once the build is successful that is build is published to the binary repository and concerned people are notified, then as the post build action deployment script should be executed. Once deployment is successful an automated testing suite can be executed to validate the application based on the testing requirement of the application.

V. CASE STUDY

A web based application is developed using java as the programming language. The application consist of eleven major modules, each module performs specific task related to the application requirement. Application is running inside a web container here using Tomcat. Apache tomcat is installed on the local servers of the environment in four instances. For the build a specific build server is configured as per the proposed architecture. On a different machine source configuration management server is installed. Perforce is used as the source configuration management tool. [7] [8] Jenkins is a continuous integration tool used as a build tool [6]. Apache Ant and maven are used for resolving dependency during build and for archiving. JFrog artifactory is used as a binary repository management tool. [10]

Once the build is completed a tar.gz is published to the artifactory. The tar file has 10 jar files and one war file packaged. The war file will be deployed inside the web container i.e. tomcat. To start the deployment shell scripts [18] are written which will run on remote server and on local servers. For every deployment steps a local script is written in the local server. Remote scripts are written to call the local scripts. The work of remote scripts is to actually control the frequency and parallelism in the process. As per the argument remote scripts gets server name, file name, instance of a service, local user name, server volume, etc from the configuration file of the environment.

VI. EXPERIMENTAL RESULTS

A. Terms

Table I: Descriptions corresponding to symbols and parameters used in equations

PARAMETER	DESCRIPTION
N	number of services to be deployed
t_d	time taken to download the binary
t_m	time taken to unmount and mount a service
t_s	time taken to restart a service on one server
k_1	amount of overhead created to access local server from remote
k_2	amount of overhead created due to parallelism

Table II: Description of a service running on how many servers

SERVICE NAME	NO OF SERVERS SERVICE RUNNING ON
Service1	c1
Service2	c2
Service3	c3
:	:
:	:
ServiceN-1	cn-1
ServiceN	cn

B. Measure

Let $P = c1 + c2 + c3 + ... + cn$ (1)

P is the total number of instances running on the local servers. A server can have two or more instance running of same or different service running on the same server.

$$P = \sum_{i=1}^{N} c_i \qquad (2)$$

To deploy an application remote server takes time T.

$$T = t_d + N(\mathrm{k}_1 + \mathrm{t}_m) + \mathrm{NP}(\mathrm{k}_1 + \mathrm{t}_s) \qquad (3)$$

Due to parallelism introduced in deployment, deployment time reduces as shown in equation 4.

$$T_p = t_d + (\mathrm{k}_1 + \mathrm{t}_m) + (\mathrm{k}_1 + \mathrm{t}_s) + \mathrm{k}_2 \qquad (4)$$

Where k_1 and k_2 are constants. For a large application, k_2 is small value as compared to N and P. For large application equation 4 can be generalized as shown in equation 5.

$$T_p = t_d + \mathrm{t}_m + \mathrm{t}_s + \mathrm{k}_2 \qquad (5)$$

C. Results

Figure 8 shows the comparison between total deployment time in second manual deployment, automated deployment and automated deployment with parallelism.

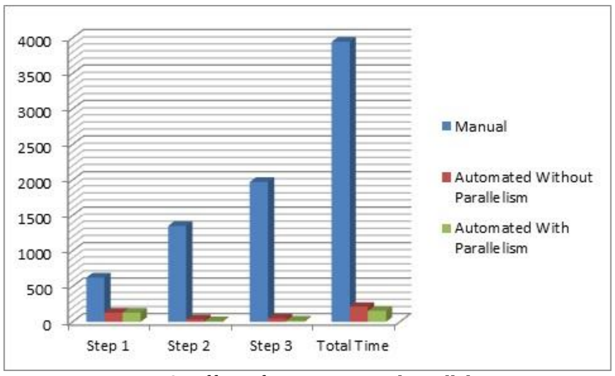


Figure 8: Effect of automation and parallelism

Figure 9 shows the comparison of deployment time in seconds for three different cases due to deployment perquisite of having a common mount point of all servers in an environment. Deployment time taken in case of full deployment, of a build version w.x.y.k and in case of deploying a service of another version is almost same. But again time to deploy another service of the same version is very less because of the proposed solution to have a common volume of all the servers in an environment. This eliminates the step to download again which takes the maximum time as build file are generally very big for an application.

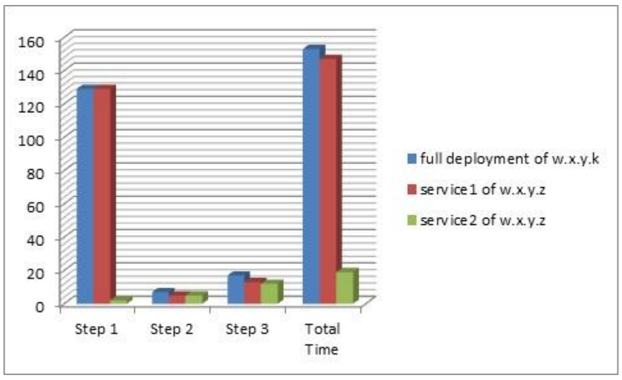


Figure 9: Effect of frequency reduction

VII. CONCLUSION

The main objective of this paper was to propose a solution to efficiently automate the whole build and deployment process. The proposed build architecture follows the best practices and also introduces the types of tools required. Deployment architecture has given generalized steps to be executed which can be executed parallely and also reduces the frequency of execution to reduce the deployment time. Proposed methodology has been successfully implemented and goal is achieved. Figure 8 and 9 shows the result of the implementation.

REFERENCES

[1] Software development environments, http://smoothtesting .blogspot.in/2011/09/different-nvironments.html.
[2] Software build, http://en.wikipedia.org/wiki/Softwarebuild.
[3] Build automation, http://en.wikipedia.org/wiki/Build _automation.
[4] Jez Humble, and Rolf Rusell, The agile maturity model applied to building and releasing software, September 2009.
[5] Martin R. Bakal, Jennifer Althouse, and Paridhi Verma, Continuous integration in agile development, August 2012.
[6] Build Tool – Jenkins, https://wiki.jenkins-ci.org/display/ JENKINS/Meet+Jenkins.
[7] Perforce, http://www.perforce.com/product/perforce
[8] Source configuration management tool comparison between subversion & perforce, http://www.perforce.com /sites/default/files/pdf/perforce-subversion-comparison.pdf
[9] Why a build server is required, http://blog.bengarney.com /2012/09/01/some-thoughts-on-build-servers/
[10] Binary repository management tool – Jfrog Artifactory, http://www.jfrog.com/home/v_artifactory_opensource_oveview
[11] Software deployment, http://en.wikipedia.org/wiki/ Software_deployment
[12] Understand linux shell and basic shell scripting languages, http://www.tecmint.com/understand-linux-shell-and-basic-shell-scripting-language-tips/
[13] O. Bushehrian, Automatic object deployment for software performance enhancement, IET Softw., 2011, Vol. 5, Iss. 4, pp. 375–384, June 2011.
[14] Tomas Kucera, Petr Hnetynka, and Jan Kofron, Automated Deployment of Hierarchical Components, R. Lee (Ed.): Software Engineering, Artificial Intelligence, Networking, SCI 443, pp. 117–128, 2013.
[15] International Technical Support Organization, The Software Deployment Mystery – Solved, August 2004.
[16] Takoua Abdellatif, Didier Hoareau, and Yves Maheo, Automated Deployment of Enterprise Systems in Large-Scale Environments, OTM Workshops 2006, LNCS 4277, pp. 30–31, 2006.
[17] Introduction to Linux, http://www.linux.org/
[18] Bash reference manual, http://www.gnu.org/software/ bash/manual/bashref.html