Walchand College of Engineering, Sangli

(An Autonomous Institute)

Department of Computer Science and Engineering

A Project Report

on

DevOps for Cloud Computing Environment

Under the Guidance of

Prof. A. R. Surve
Guide
Professor, Computer Science & Engg Dept,
WCE, Sangli

Mr. Prasanna Kulkarni
Co-Guide
Senior Manager,
Veritas Technologies LLC, Pune

Submitted by

Vaibhav Ananda Kumbhar

2012BCS057

2015-2016

Walchand College of Engineering, Sangli
(An Autonomous Institute)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the B.Tech. Project entitled **DevOps for Cloud Computing Environment** is a bonafide work carried out by **Mr. Vaibhav Ananda Kumbhar** in partial fulfillment of the completion of B. Tech. project during the year 2015-2016. The project report has been approved as it satisfies the academic requirement with respect to the project work.

Prof. A. R. Surve	**Mr. Prasanna Kulkarni**	**Prof. Dr. B. F. Momin**
Guide	Co-Guide	HOD
Professor, CSE Dept,	Senior Manager, Veritas	CSE Dept,
WCE, Sangli	Technologies LLC, Pune	WCE, Sangli

Acknowledgement

I wish to take this opportunity to express my sincere gratitude to all the people who have extended their cooperation in various ways during my project work. It's my pleasure to acknowledge all those individuals.

I wish to express my sincere gratitude to Mr. Prasanna Kulkarni, Co-guide, Senior manager, and Mr. Aatish Arora, Senior Manager, for providing me an opportunity to do my internship and project work in "VERITAS TECHNOLOGIES, LLC". With their guidance, cooperation and encouragement, I learnt various new things during our project tenure.

I would like to thank my project guide Prof. A. R. Surve sir, Computer Science and engineering Department for his guidance and help throughout the development of this project work by providing me with required information.

I would like to thank all my team members in VERITAS who helped me in every aspect to complete this project successfully.

I specially thank Dr. B. F. Momin Head, Computer Science and Engineering Department for his continuous encouragement and valuable guidance in bringing shape to this dissertation.

I specially thank Dr. G.V. Parishwad, Director, of Walchand College of Engineering, Sangli for his encouragement and support.

Declaration

I hereby declare that the project work entitled **"DevOps for Cloud Computing Environment"** submitted to the Walchand College of Engineering, Sangli is a record of an original work done by me under the guidance of **Prof. A. R. Surve, Computer Science & Engineering** and **Mr. Prasanna Kulkarni, Senior Manager, VERITAS Technologies LLC.,Pune** and this project work is submitted in the partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science & Engineering.

Date:

Place:

Vaibhav A. Kumbhar
2012BCS057,
B.Tech. C. S. E.

Abstract

Today the IT development is growing very fast. Customer needs updates of product more frequently. The code quality should be good so that improvement and bug fixing in the product will be more easily and quickly.

The purpose of **DevOps for Cloud Computing Environment** is to do software development fast and seamlessly by minimizing the gap between developers teams and other IT operations teams. Veritas Resiliency Platform (VRP) will be able to release products more frequently.

In this project's context, we focuses on the source code analysis. The source code should be well tested so that it will eventually improves code quality.

Contents

List of Figures

List of Acronyms

CI/CD Continuous Integration and Continuous Delivery

CI Continuous Integration

CD Continuous Delivery

VRP Veritas Resiliency Platform

POM Project Object Model

JaCoCo Java Code Coverage

DC Data Center

VCS Version Control System

SCM Software Configuration Management

LOC Lines of Code

KLOC Thousands lines of code

IDE Integrated Development Environment

VSA Virtual Service Appliance

Chapter 1

INTRODUCTION

1.1 Introduction to DevOps

What is DevOps? DevOps is a culture that company can use it to develop big projects seamlessly to achieve fast development and delivery of products to customer by collaborating developers and operations team together.

A few decades back, almost all systems were standalone. But nowadays, everybody wants access to application from anywhere at any time. That's why web applications became more popular. Instead of dedicated servers, in today's era, cloud computing is becoming more popular and it is where shared resources, data and information are provided to computers and other devices on-demand.

As cloud computing environments are distributed and virtualization needs to be extensively used, it is more complex to develop a cloud based solution. We need to implement DevOps for cloud computing environment so that the development of cloud based solutions will become easier. We can help operations and development engineers participating together in the entire service lifecycle, from design through the development process to production support.

Scope of DevOps includes phases from planning to the actual deployment of product as shown in figure 1.1. DevOps tries to cover the delivery flow of product from phase planning to deployment. DevOps ensures that the Continuous Integration and Continuous Delivery has been achieved.

1.2 Introduction to Cloud Computing Environment

Cloud computing is single point of interaction to user with distributed hardware resources. Virtualization technologies are used by cloud computing environment. Openstack is one open

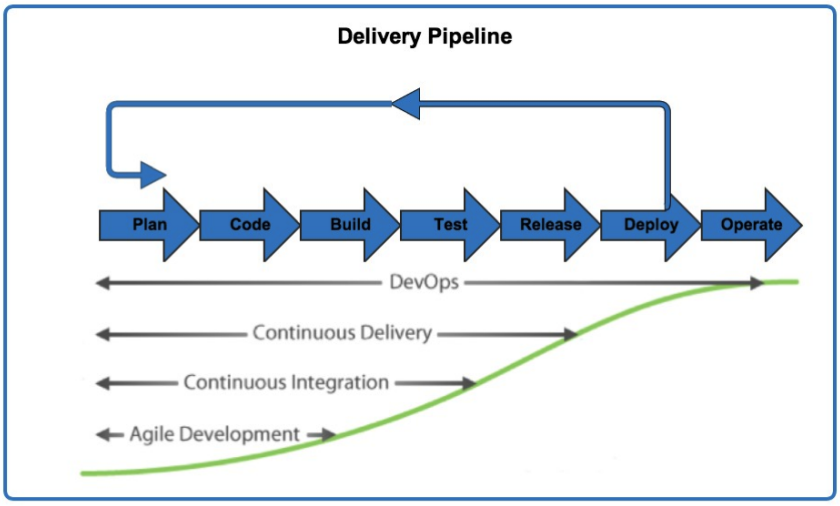


Figure 1.1: Delivery Pipeline [1]

source software to enable cloud technology.

Main two types of cloud are as following: Public cloud, private cloud. Private clouds are installed by their own organization. Public clouds, like AWS, etc., can be used on the basis of resource usage and cloud service provider/cloud vendor charge for it.

1.3 Introduction to VRP

Veritas Resiliency Platform (VRP) provides resiliency to virtual machines in the data centers by using cloud data center as secondary/recovery data center. The various virtualization technologies like Hyper-V and VMware supported by VRP for resiliency. The cloud data centers used as secondary data center for recovery in case of disaster. Openstack cloud technology supported for secondary data centers.

VRP handles these various technologies, so to integrate them altogether and to develop product fast and seamlessly, to implement DevOps was necessary.

1.4 Continuous Integration and Continuous Delivery

Continuous Integration (CI) is a approach of integrating all working copies of developers on central shared mainline without causing failures to each others code. Developers can be able see how their change in existing or new code will behave into the actual product.

Continuous Delivery (CD) is a approach in which teams produce software in short cycles. Each cycle ensures that we can release high quality software fast through build, test and deployment automation. The benefits we can get from this is that it reduces the cost, time, and risk

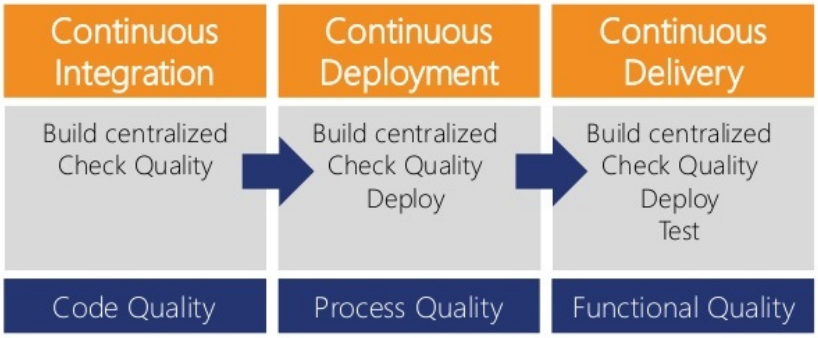


Figure 1.2: Continuous Integration vs Continuous Delivery vs Continuous Deployment [2]

of delivering changes. We will be able to do incremental updates easily and more frequently. Agile methodology is best suited to achieve continuous delivery(CD). Development life cycle is divided into number of short cycles known as sprints. At the end of each sprint every story has to be completed so that after every sprint release team can be able to release product.

"Continuous Integration" ensures code quality by checking compatibility with another's code. "Continuous Delivery" ensures the functional quality whether it is can be deployable or not. It checks only proper functionality, It does not deploys product anymore. "Continuous Deployment" ensures the process quality by test, automation acceptance testing and then product deploys automatically. Relevance between these three terms is shown in figure 1.2

Chapter 2

TECHNICAL SPECIFICATIONS

2.1 Project Management

Maven is software project management tool. The projects distributed among various team working on various technologies can be dependent on each other. Maven is best suited to develop products in multi module environment projects. Maven is based on declarative model. It defines Project Object Model (POM) in pom.xml file

2.1.1 Maven Operations

Maven can do following operations:

1. Builds: Makes build process easy. It takes source code, compiles it and packages them into executables.
2. Reporting: It generates reports as well which includes build log, used third party plugins in project.
3. Documentation: Keeps project description, summary.
4. Dependencies: Keeps track of third party plugin dependencies. And it keeps track of dependencies between child modules in same project.
5. Software Configuration Management (SCM)s: Maven can push code automatically to version control system if the build succeed and it also includes the recent changes done with respect to previous build and who did that changes.

6. Releases: Maven handles the release management by using project versions and update to next version. Maven can rollback to previous version.

7. Distribution: Maven puts repositories on server. Anyone can download latest build/executable from there.

8. Mailing list: Maven puts mailing list on site to subscribe/unsubscribe. Email will be broadcast to all members.

2.1.2 Maven Lifecycle

Maven works around build cycle. Following are the maven phases of lifecycle:

2.1.2.1 Clean Lifecycle

Clean lifecycle removes the files and folders created by previous build. Phases in clean lifecycle are as following:

1. pre-clean: execute processes needed prior to the actual project cleaning

2. clean: remove all files generated by the previous build

3. post-clean: execute processes needed to finalize the project cleaning

2.1.2.2 Build Lifecycle

Build lifecycle actual produces software deliverables like .jar or .exe files. Phases in build lifecycle are as following:

1. validate: validate the project is correct and all necessary information is available.

2. initialize: initialize build state, e.g. set properties or create directories.

3. generate-sources: generate any source code for inclusion in compilation.

4. process-sources: process the source code, for example to filter any values.

5. generate-resources: generate resources for inclusion in the package. It copies resources from the resources directory into the output directory to include it into the installable file like jar.

6. process-resources: copy and process the resources into the destination directory, ready for packaging.

7. compile: compile the source code of the project.

8. process-classes: post-process the generated files from compilation, for example to do byte-code enhancement on Java classes.

9. generate-test-sources: generate any test source code for inclusion in compilation.

10. process-test-sources: process the test source code, for example to filter any values.

11. generate-test-resources: create resources for testing. It generates the resources required while testing in test resources directory.

12. process-test-resources: copy and process the resources into the test destination directory.

13. test-compile: compile the test source code into the test destination directory

14. process-test-classes: post-process the generated files from test compilation, for example to do bytecode enhancement on Java classes. For Maven 2.0.5 and above.

15. test: run tests using a suitable unit testing framework. These tests should not require the code be packaged or deployed.

16. prepare-package: perform any operations necessary to prepare a package before the actual packaging. This often results in an unpacked, processed version of the package. (Maven 2.1 and above)

17. package: take the compiled code and package it in its distributable format, such as a JAR.

18. pre-integration-test: perform actions required before integration tests are executed. This may involve things such as setting up the required environment.

19. integration-test: process and deploy the package if necessary into an environment where integration tests can be run.

20. post-integration-test: perform actions required after integration tests have been executed. This may including cleaning up the environment.

21. verify: run any checks to verify the package is valid and meets quality criteria.

22. install: install the package into the local repository, for use as a dependency in other projects locally.

23. deploy: done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects.

2.1.2.3 Site Lifecycle

Site lifecycle generates site for the project regarding build information. Site includes information about the third party used plugins, artifacts of current build version, etc. Phases in site lifecycle are as following:

1. pre-site: execute processes needed prior to the actual project site generation
2. site: generate the project's site documentation
3. post-site: execute processes needed to finalize the site generation, and to prepare for site deployment
4. site-deploy: deploy the generated site documentation to the specified web server

2.1.3 Maven Profile

In Maven, we can define specific configuration in profile. When that profile activates then the executions defined in that profile runs. We can define multiple profiles for different goals. Profile has multiple goals which are associated with any phase of lifecycle.

For ex.: To clean the directory other than default output directories, needs to be delete in clean phase of clean lifecycle. To delete that directory, we could specify goal. We can activate one or more profiles at a time.

2.2 Continuous Integration

Jenkins is a automation server. Professionals uses it to build software products. Once Jenkins configuration is ready, then it downloads source code from central repositories like stash, git hub, bit bucket, etc. and builds the software.

Jenkins can schedule periodic build. Several build are fired in one day so that the developers can test their code latest build.

2.3 Continuous Delivery

Agile methodology is best to achieve continuous delivery. JIRA is project development and issue tracking tool. At the beginning of every sprint, a story has been decided first then actual development starts. Once story decides then it is divided into small tasks and for every task incident on JIRA created. At the end of sprint all JIRA incidents should be resolved and completed without any defect.

2.4 Code Coverage

What is Code Coverage? The Code Coverage means to check whether the code written by developer is well tested or not. JaCoCo is java code coverage tool which instruments java code and determines coverage matrices. JaCoCo is best because it does the instrumentation on-the-fly while unit testing, so there is no need to do offline instrumentation. On-the-fly instrumentation reduces overall time to build and analysis. [3]

2.4.1 Code Coverage Counters

Units to measure code coverage by various techniques depending on different behaviors of code elements.

1. **Instruction Coverage**: Typically it is related to the java byte code instructions. This counter indicates percentage of instructions covered by unit testing.
2. **Branch Coverage**: Includes the IF-ELSE, SWITCH, etc code element which lead to branching. Source Code Highlighters:
   - Red diamond: No coverage.
   - Yellow diamond: Partial coverage. Some branches covered.
   - Green diamond: Full coverage.
3. **Line Coverage**: This column depicts how many lines in source code has been covered by unit testing. Line counter unit is Lines of Code (LOC)/Thousands lines of code (KLOC).
4. **Method Coverage**: Coverage percentage of method covered in unit testing. If any branch inside the method is uncovered then the method will be marked as uncovered.
5. **Class Coverage**: The class has been instantiated or not determines class coverage. If the class is not instantiated in unit testing, then class will be marked as uncovered.

2.4.2 Cyclomatic Complexity

Number of linearly independent paths in source code. It determines number of test cases required to cover all source code. for ex:

- **If-else**: Only one section will execute either if block or else block thats why complexity would be 1.
- Only **IF** condition no else section: When *IF* condition becomes true that would be one path and if the *IF* condition false that would be another path so complexity would be 2.

- If **IF** condition with two condition then complexity would be 2.

Chapter 3

SYSTEM ANALYSIS

3.1 Overview

All developers, QA team and other IT operations teams analyzes unit test report and makes decision to accept code for further testing. All developers will be able to analyze their own code locally.

3.2 Proposed System

DevOps is partially implemented for VRP. One Jenkins project to do coverage analysis and publishes results to all teams.

Unit testing should be done in parallel so that it will optimize execution time. Along with unit tests, system collects the coverage statistics. After complete execution the system will publish reports and sends report link to all people who are working on VRP.

3.3 Hardware Requirements

- x86 processor Desktop/Laptop.

3.4 Software Requirements

- Maven
- Eclipse
- JaCoCo
- Jenkins

Chapter 4

SYSTEM DESIGN

4.1 Architecture Diagram

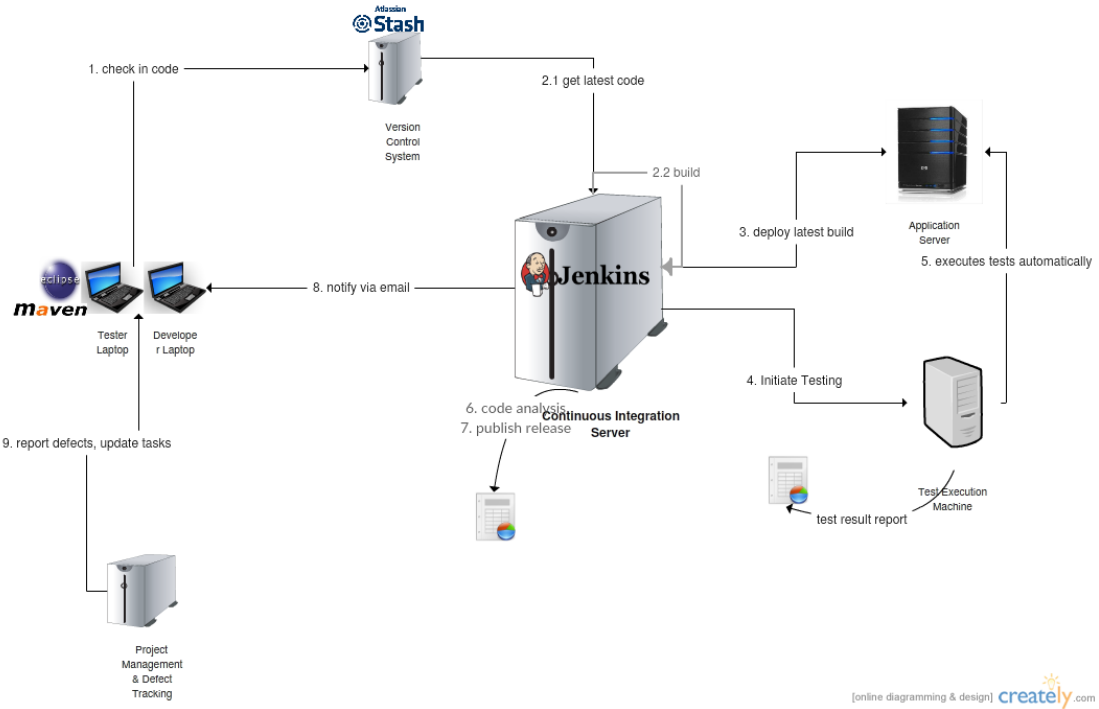


diagram.png

Figure 4.1: Architecture Diagram

DevOps system comprises of the components shown in the figure 4.1. Jenkins CI server plays vital role in DevOps. Jenkins CI server is central place to all people working in project.

4.1.1 Architecture Explanation

Data flow and sequence of activities are shown in the figure 4.1. Some more information of activities included in architecture is as following:

1. **Check in code**: Developer writes code and merges it with repository on VCS. Similarly, Tester writes test cases and merges it with repository on VCS.
2. **Build**: Jenkins CI server gets latest updated code from repository and builds the product.
   (a) Get latest code: Copies source code repository from VCS to the Jenkins workspace.
   (b) Build: Builds the product from copied source code and generates all executables.
3. **Deploy latest build**: CI server deploys latest build on application server and configures it. CI server starts software product service on application server.
4. **Initiate testing**: Once application is ready, then CI server initiates testing.
5. **Execute tests**: Test execution machine starts tests on app;ication server one by one automatically and collects log for test cases both passed and failed.
6. **Code analysis**: Code analysis starts on the source code which were used to build product.
7. **Publish release**: Once the source code analysis and the testing done application server done then publishes release reports. Reports includes the test report and code analysis report. Once reports analyzed by release team then product will be ready for release.
8. **Notify via email**: If any code broken in build phase then CI server notifies the respective developer or tester with the failure log.
9. **Report defects and update tasks**: Once product has been released then the real time defects arises from the customer. Customer gives suggestions to add new feature. Accordingly the new tasks created and given to the developer for implementation.

4.2 UML Diagrams

4.2.1 Use Case Diagram

Use cases included in DevOps are as following.

4.2.1.1 DevOps Use Case

DevOps Use Case view shown in figure 4.2. DevOps engineer is one actor who has very important role in the product. He is the one person who handles all product development

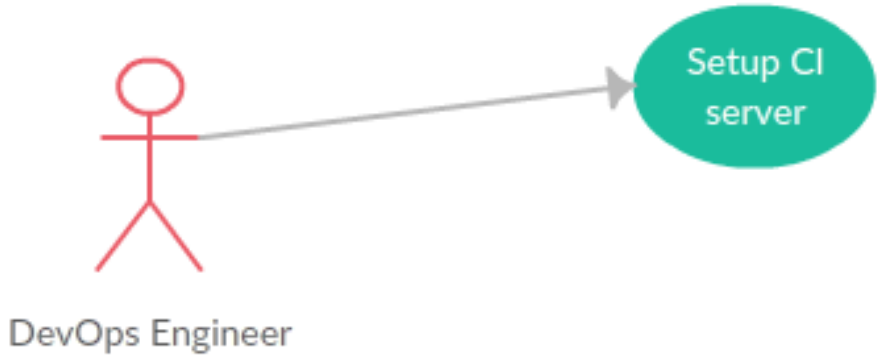


Figure 4.2: DevOps Use Case Diagram

and delivery process. DevOps engineer has to setup the DevOps system. CI server is central component in DevOps system. DevOps engineer is caller of *setup CI server* use case.

4.2.1.2 Build Management Use Case

Build Management Use Case view shown in figure 4.3. Build manager is on top of all who has all rights. If he wants to stop DevOps system he will call *stop CI server*. Similarly, he wants to start DevOps system he calls *start CI server*.

4.2.1.3 Development Use Case

Development Use Case view shown in figure 4.4. This includes two use cases one is commit code and another is build product under development environment. Developer and Tester are two actors calls these use cases. Actors playing role in this:

1. Developer: Developer wants to build product with his changes. He calls *build* under dev environment. Once he is done with correct changes then he *commits code* and pushes it central repository.

2. Tester: Tester writes test cases and builds product with his test cases. If his test cases are completed then he adds his test cases to central repository by calling commit code.

4.2.1.4 Build Use Case

Build Use Case view shown in figure 4.5. Build engineer is one actor plays role in build use case. Build engineer checks the updations in source code repository periodically by calling *check*

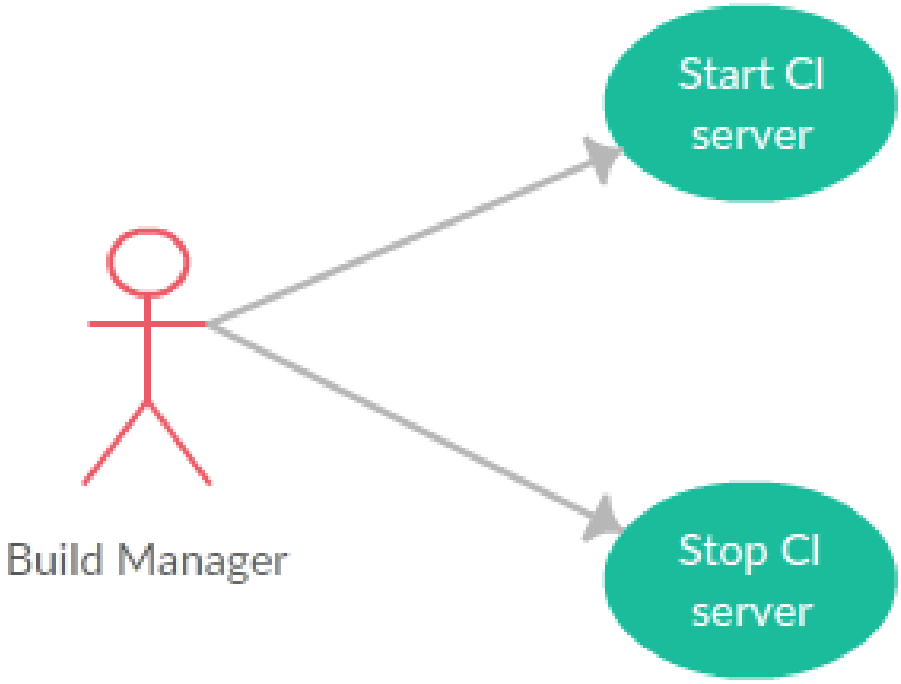


Figure 4.3: Build Management Use Case Diagram

for updates. If new changes found in latest source code, he fires *build*. *Build prod_env* use case called by build engineer, CI server via scheduled build or by any user via parameterized build.

4.2.2 Class Diagram

Class diagram shown in figure 4.6. Details of classes in class diagram.

4.2.2.1 Environment

Environment enumeration to set environment according to type of environment. Types of environment are Development, Test and production.

4.2.2.2 Buildable

This is interface defines two behaviors for classes. *init* behavior initializes environment. *fireBuild* starts build of product. Implementer of this interface has to write this behavior.

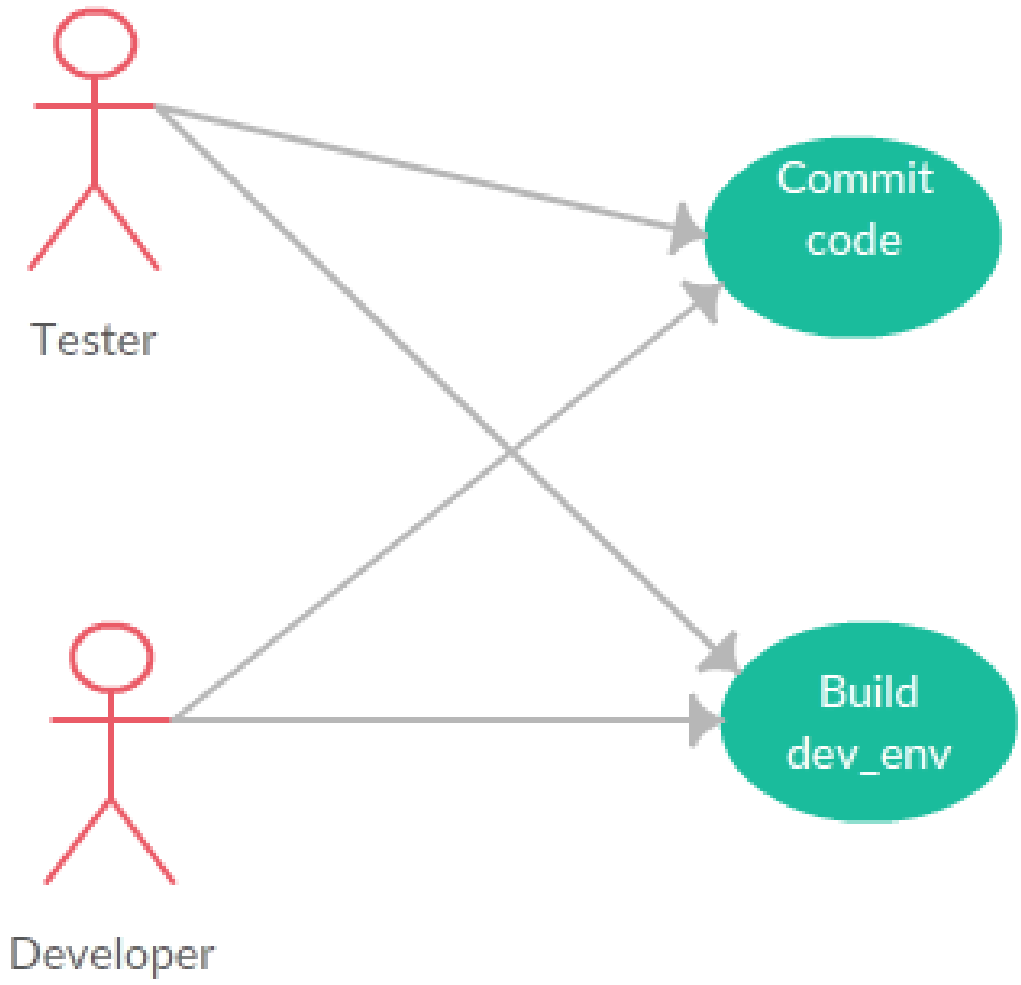


Figure 4.4: Development Use Case Diagram

4.2.2.3 Developer

Developer can *fire build* and *get log* of that build.

4.2.2.4 Tester

Tester can fire build with test environment enabled and get log of test case execution.

4.2.2.5 Build Engineer

Build engineer can fire build and get log of that build. He also can get the high level reports of that build which includes the information of successful build.

4.2.3 Sequence Diagram

Sequence of operations in DevOps as shown in figure 4.7.

Lifelines in the sequence diagram as following:

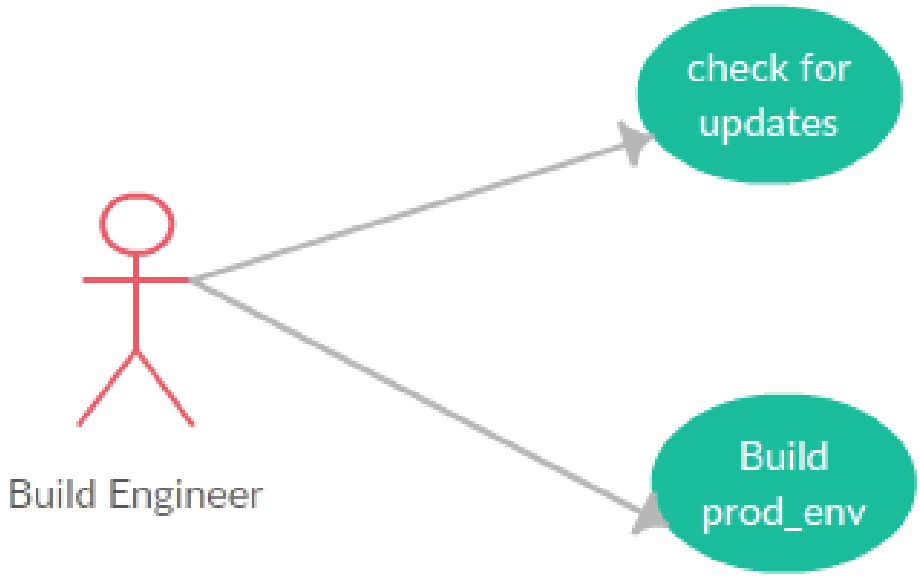


Figure 4.5: Build Use Case Diagram

4.2.3.1 Delivery Team

Delivery team is actor who is triggering all the sequence of operations.

4.2.3.2 VCS

VCS is a Entity lifeline where source code of product is available.

4.2.3.3 Build & unit tests

Lifeline from which sequence can be stopped if any failure in the build and unit testing. If builds successfully then triggers next sequence.

4.2.3.4 Automated acceptance tests

Controller lifeline, managed by test execution machine, tests the system by running automated test cases.

4.2.3.5 Release

Release is controller lifeline where product release can be declined or approved.

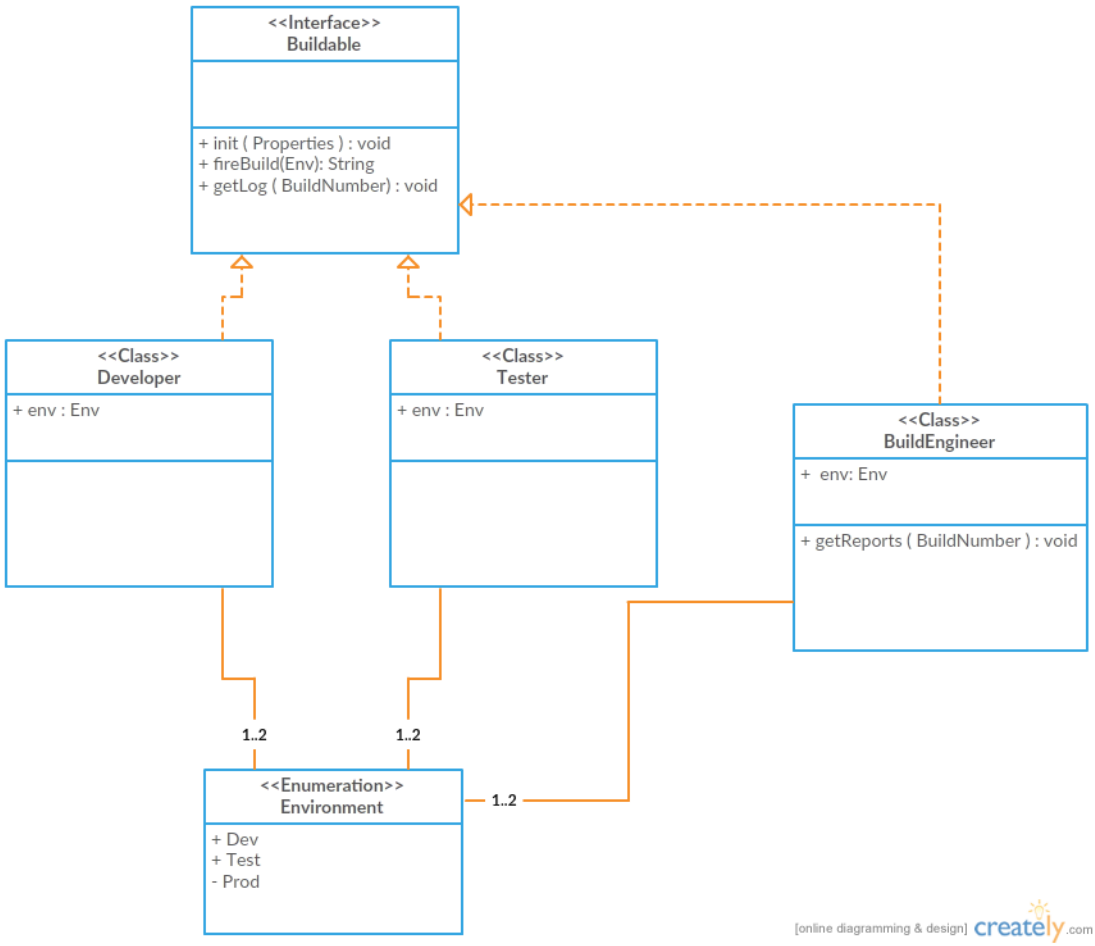


Figure 4.6: Class Diagram

4.2.3.6 Deployment

Deployment is boundary lifeline. Once this lifeline boundary reached the product will ready to release.

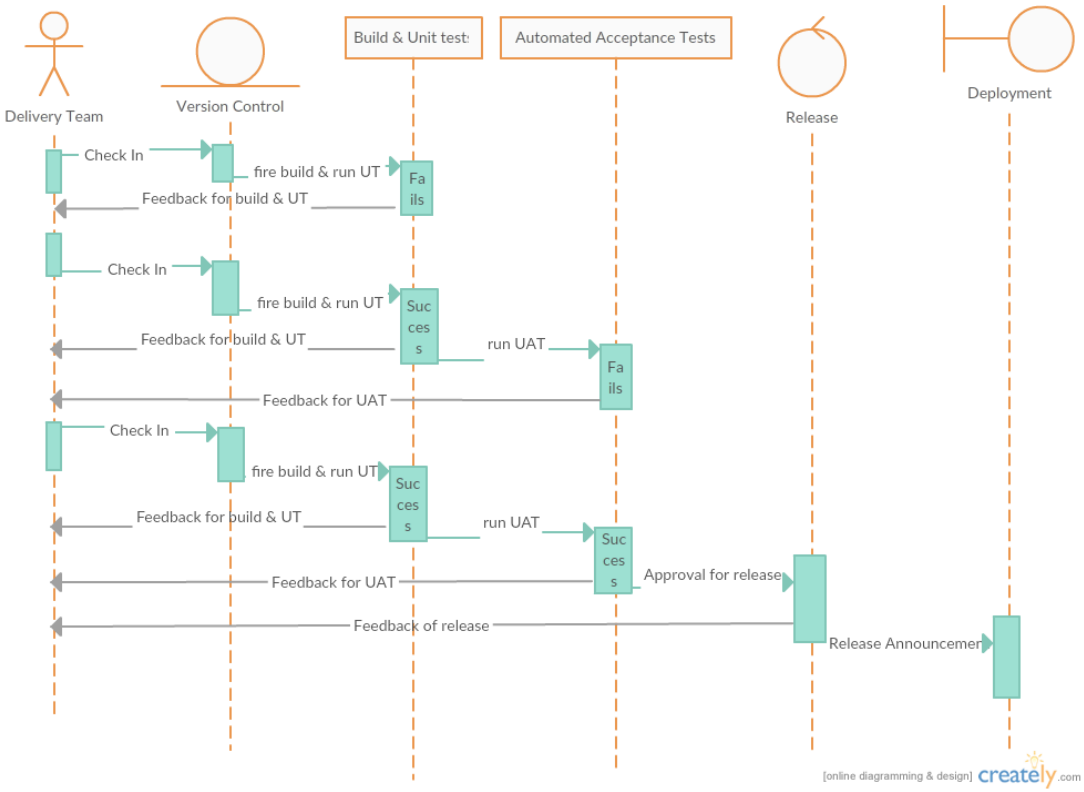


Figure 4.7: Sequence Diagram

Chapter 5

IMPLEMENTATION

5.1 Development Environment

Eclipse is open source IDE used by almost all developers. For eclipse environment defined one build configuration by enabling JaCoCo instrumentation. Integration of JaCoCo with eclipse done by writing one maven profile 2.1.3. Within profile, executions defined for prepare-agent (which does the instrumentation) and report(generates report).

5.2 Production Environment

Jenkins is used to build VRP. One Jenkins projects is created for code coverage analysis. Project publishes coverage result to all teams of VRP by email. Thresholds has been set to accept code for further testing.

Chapter 6

SYSTEM TESTING

6.1 Developer Team

Developer team uses eclipse as development environment to build product and test their written code. Developer has to fire build with unit test by enabling JaCoCo instrumentation. Once build succeed without error and test failure, code coverage report will be available in report output directory. Developer has to ensure that the collecting coverage statistics and report is correct.

6.2 Build Team

Build team uses Jenkins CI server to collect and view coverage statistics. Build team has to ensure that all builds are going in proper without harming other project or parallel build.

6.3 QA Team

QA team ensures that the functional testing code coverage is up to the mark to consider it for further testing.

Chapter 7

SCREENSHOTS AND RESULTS

7.1 Screenshots

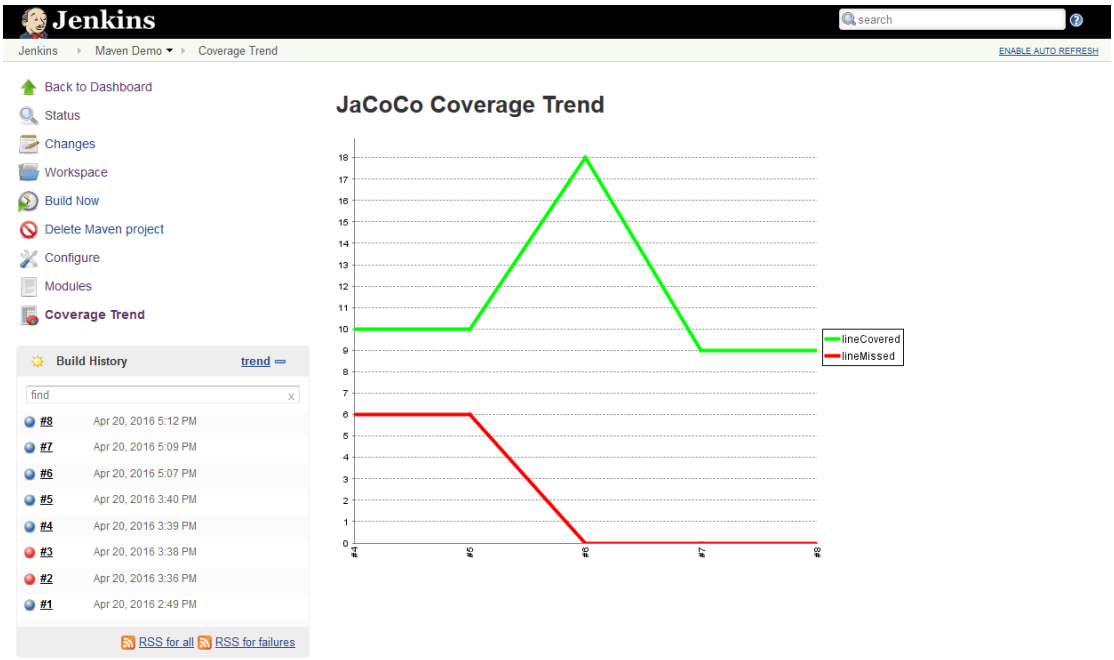
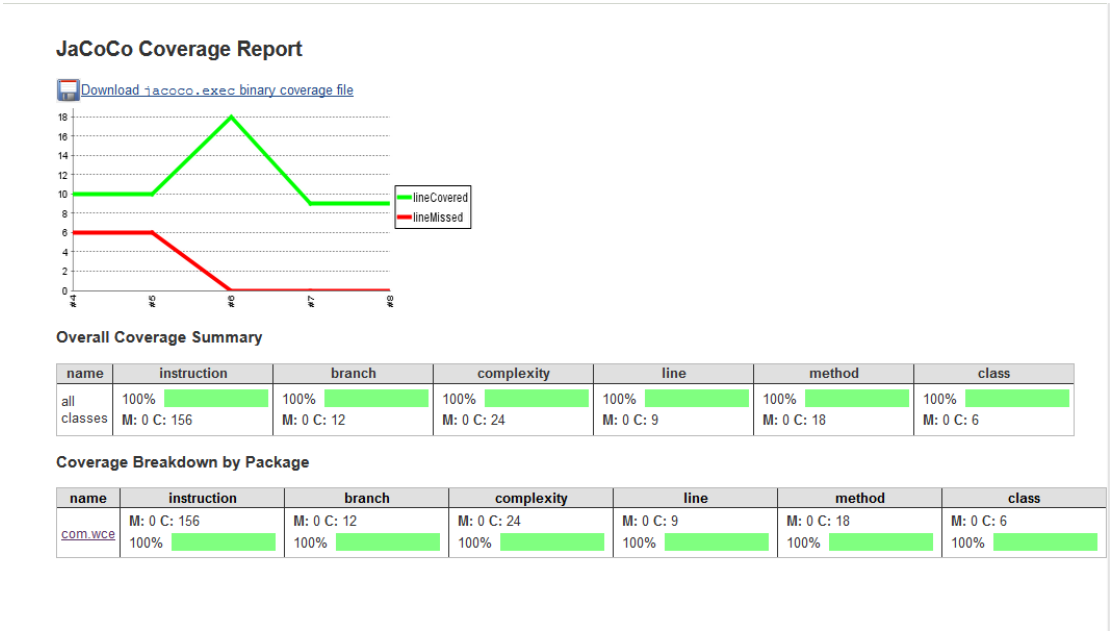


Figure 7.1: Code Coverage Trend

7.2 Result

Eclipse and Jenkins project is doing code analysis. Jenkins builds daily and publishes report to all teams.



JaCoCo Coverage Report

Download jacoco.exec binary coverage file

lineCovered
lineMissed

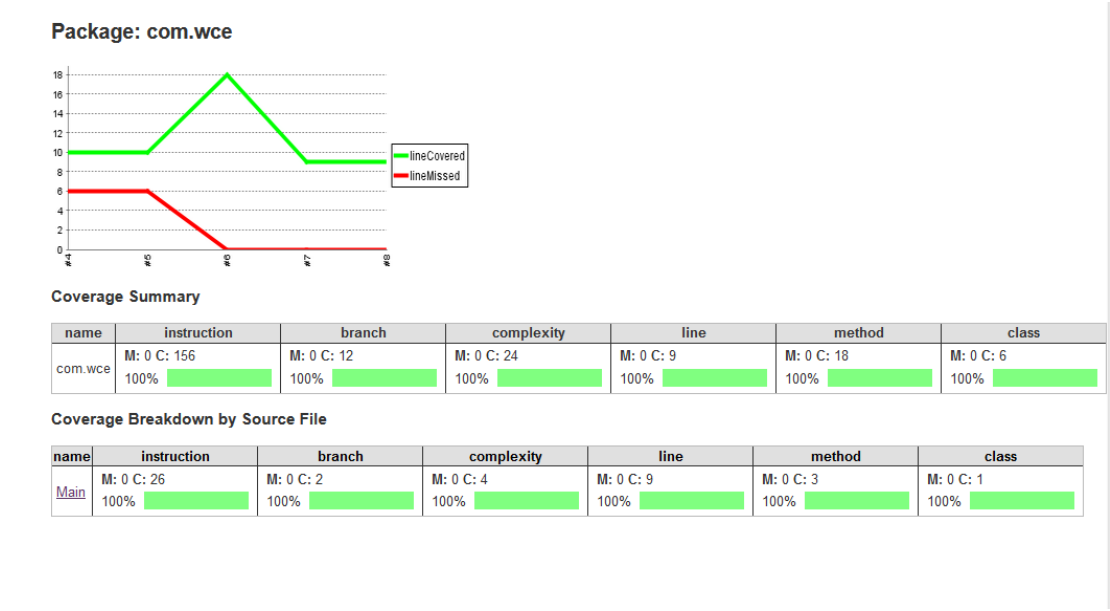
Overall Coverage Summary

name	instruction	branch	complexity	line	method	class
all classes	100% M: 0 C: 156	100% M: 0 C: 12	100% M: 0 C: 24	100% M: 0 C: 9	100% M: 0 C: 18	100% M: 0 C: 6

Coverage Breakdown by Package

name	instruction	branch	complexity	line	method	class
com.wce	M: 0 C: 156 100%	M: 0 C: 12 100%	M: 0 C: 24 100%	M: 0 C: 9 100%	M: 0 C: 18 100%	M: 0 C: 6 100%

Figure 7.2: Code Coverage Report



Package: com.wce

lineCovered
lineMissed

Coverage Summary

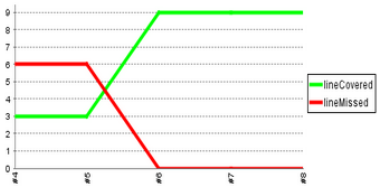
name	instruction	branch	complexity	line	method	class
com.wce	M: 0 C: 156 100%	M: 0 C: 12 100%	M: 0 C: 24 100%	M: 0 C: 9 100%	M: 0 C: 18 100%	M: 0 C: 6 100%

Coverage Breakdown by Source File

name	instruction	branch	complexity	line	method	class
Main	M: 0 C: 26 100%	M: 0 C: 2 100%	M: 0 C: 4 100%	M: 0 C: 9 100%	M: 0 C: 3 100%	M: 0 C: 1 100%

Figure 7.3: Package Coverage

Package: Main



Main

name	instruction	branch	complexity	line	method
Main()	M: 0 C: 3 100%	M: 0 C: 0 0%	M: 0 C: 1 100%	M: 0 C: 1 100%	M: 0 C: 1 100%
main(String[])	M: 0 C: 14 100%	M: 0 C: 0 0%	M: 0 C: 1 100%	M: 0 C: 5 100%	M: 0 C: 1 100%
method1(int)	M: 0 C: 9 100%	M: 0 C: 2 100%	M: 0 C: 2 100%	M: 0 C: 3 100%	M: 0 C: 1 100%

Coverage

```
1: package com.wce;
2:
3: public class Main {
4:     public static int method1(int data) {
5:•        if (data == 1) {
6:            return 0;
7:        } else {
8:            return data+1;
9:        }
10:    }
11:    public static void main(String[] args) {
12:        System.out.println("Main Method");
13:        method1(1);
14:        method1(2);
15:        Main main= new Main();
16:    }
17: }
```

Figure 7.4: Class Coverage

Chapter 8

CONCLUSION AND FUTURE WORK

All teams working on VRP are getting coverage analysis regularly. VRP is increasing code quality because all developer are writing unit tests to cover all code they have written.

This project can be extend to find code vulnerabilities in the source code.

Bibliography

[1] "Delivery pipeline," 2014. [Online]. Available: http://cdn.panthacorp.com/wp-content/uploads/2014/04/Image-3.png

[2] "Continuous integration vs continuous delivery vs continuous deployment," 2014. [Online]. Available: http://image.slidesharecdn.com/sharepointcontinuousintegrationwithvsonlineandazure-140510101655-phpapp01/95/sharepoint-continuous-integration-with-vsonline-azure-espc14-4-638.jpg?cb=1399949644

[3] (2016) Code coverage metrices. [Online]. Available: http://eclemma.org/jacoco/trunk/doc/counters.html

[4] P. Labs, *2015 State of DevOps Report*, 2015.

[5] S. Karadzhov, *Fundamentals of Continuous Integration*, 2014.

[6] DevOps.org. (2016, Jan.) Devops. [Online]. Available: http://devops.com/

[7] Owasp.org. (2016, Jan.) Static code analysis. [Online]. Available: https://www.owasp.org/index.php/Static_Code_Analysis

[8] Apache.org. (2016, Jan.) Maven. [Online]. Available: http://maven.apache.org/

[9] J. Blog. (2016, Jan.) Open source automation server. [Online]. Available: https://jenkins-ci.org/node/

[10] Wikipedia. (2016, Jan.) Sanity tests. [Online]. Available: https://en.wikipedia.org/wiki/Sanity_check

[11] C. Inc. (2016, Jan.) Coverity. [Online]. Available: http://www.coverity.com/products/coverity-save/

[12] P. Lab. (2016, Jan.) Ci/cd. [Online]. Available: https://puppetlabs.com/blog/continuous-delivery-vs-continuous-deployment-whats-diff

[13] Jenkins jacoco. [Online]. Available: https://wiki.jenkins-ci.org/display/JENKINS/JaCoCo+Plugin

[14] (2014) Jenkins plugins. [Online]. Available: http://www.hugeinc.com/ideas/perspective/list-of-useful-jenkins-plugins

[15] (2016) Devops docker. [Online]. Available: https://www.docker.com/use-cases/devops

[16] (2010) History of devops. [Online]. Available: http://www.infoq.com/news/2010/04/debate-role-of-ops

[17] (2010) Continuous delivery vs deployment. [Online]. Available: http://continuousdelivery.com/2010/08/continuous-delivery-vs-continuous-deployment/

[18] Jacoco code coverage library. [Online]. Available: https://github.com/jacoco/jacoco/tree/agent-debug-logging

[19] Maven introduction. [Online]. Available: https://www.waltercedric.com/index.php?option=com_content&view=article&id=1795&catid=129&Itemid=332

[20] (2012) Jenkins installation. [Online]. Available: http://sonar-jenkins.blogspot.in/2012/05/setup-jenkins-in-local-machine.html

[21] "Continuous delivery vs continuous deployment vs continuous integration," 2012. [Online]. Available: https://blog.assembla.com/AssemblaBlog/tabid/12618/bid/92411/Continuous-Delivery-vs-Continuous-Deployment-vs-Continuous-Integration-Wait-huh.aspx