Codmon-VM: A multi-platform modular test environment.

Berend van Veenendaal

April 13, 2014

Abstract

TODO:Abstract

Preface

TODO: Preface,acknowledgements

Contents

1 Introduction

This chapter introduces the Codmon-VM project by giving a brief description of background of my research and of the previous version of the Codmon project. It also describes the structure of this thesis.

1.1 Background

In times when software projects become more and more complex, testing of this software becomes more and more important. Many software related problems are caused by lack of testing of the software [17]. Most of the times software testers only test software on one platform. For instance only on a Linux or a Windows platform. Setting-up and configuring again and again all the different test environments on different platforms simply costs too much time. So one of the challenges of software testing is to make sure that the software behaves in the same way on different platforms, without spending too much time on the installation and configuration of the test environment on all these platforms. Even when the test environment is written in such a way that it is able to run on different platforms, there are still issues that must be dealt with, before one is able to run and test the software. So in an ideal world we can test the software without being worried about setting up the test environment.

1.2 Problem indication

Nowdays there are numerous test frameworks and test environments available. For example there is *Junit*[9] for Java-unit testing and *NUnit*[12] for C#-unit testing. There are also different environments like Hudson[4][13], Jenkins[5] which can build a project and run a series of (unit) tests against this project. These frameworks and environments have both their advantages and disadvantages. One of the advantages of unit testing is that a software developer can easily add new *functional* unit tests. One of the disadvantages is that standard unit testing ignores non-functional tests like performance testing and the deployment of the software. Jenkins and Hudson,like Unit tests, also have their disadvantages. For instance, although they both run on several platforms, in their usage they are not really platform independent. For example, if you want to make a connection from Hudson or Jenkins to a remote machine you do this by executing a shell script, so a user must know in advance on which platform this script has to run. So although it is possible to connect to different machines it is not a 100% platform independent environment.

1.3 Problem statement

The test frameworks and test environments mentioned in section 1.2 can be criticized on one or more aspects. What we are looking for is in fact, a combination of the positive aspects of the described frameworks and environments, without the undesirable aspects. So the central question is, is it possible to design a multi-platform, modular test environment? In addition, we study if it is possible to design the test environment in a *user friendly* way, meaning that it must be possible to easily add both new test cases and software without knowing anything about the internal mechanisms of the test environment.

This thesis describes a multi-platform, user friendly modular test environment called Codmon-VM. The Codmon-VM project provides users with a set of virtual machines, in which Codmon is already installed and preconfigured. The purpose of the virtual machines is to make it easy for users to test software in several environments. When a user wants to test his software on Windows 7, he should download the Codmon-VM windows 7 VM and install it in VM-ware or Virtual box. The same applies for testing his software in an Ubuntu environment. By doing it this way the only tasks a user of Codmon-VM has to do are 1) add their project to an initialization file and 2) add the tests to a so called wrapper file. This will be discussed in more detail in section 4.

1.4 Thesis outline

Section 2 first describes the original Codmon framework and why it was built. It also identifies the problems it has. In section 3, *The road to Codmon-VM*, we explain how we got to the final Codmon-VM design. Section 4 describes the implementation of the Codmon-VM project. It starts with a general description of the project followed by a detailed explanation of the different modules of Codmon-VM. After this we will evaluate the choices and their consequences in section 5. In Section 6 we discuss the results based on section 5. We end this section with a brief discussion of related work.

2 Codmon

In this section we describe the original Codmon framework and its shortcomings. The original Codmon framework was built in 2005 by François Lesuer[10]. Originally Codmon was built for testing and performance monitoring Ibis projects[14][15][11][18][7] on the DAS-2[3] computer. Codmon was able to perform both functional and performance tests. [1] If for some reason a particular test failed, Codmon raised an alarm and reported the failures. Codmon does this by sending an email directly to the programmer who made the last changes in the software that was tested. Codmon also reported in the same way in case the performance drops below a certain threshold. Next to sending an email in case of failing tests, Codmon also marks them on a results web page. It was also the intention that Codmon would be extensible. In this the Codmon programmers succeeded only partially. We will discuss this more in depth in section 2.2.

2.1 The working of Codmon

In this section we will describe the design of the Codmon framework. The Codmon framework is more or less modular and consists of a two parts. The first part is the actual Codmon program. This part is responsible for a few things, which we will explain in a few minutes. When a user wants to test one of the applications mentioned above, the only thing he or she must do is to make sure that the Codmon framework is installed on the Das-2. He or she needs to know nothing of the Codmon program, except for how it should be started. This *core-part* of Codmon is responsible for a few things. first it creates a *shot* of the actual state of the of the programs that are tested. It does this by executing a set of *sensors* and collecting the output of these executions. later in this section we'll explain what sensors are. In case of a performance tests, the sensor file will make the distinction between functional test and performance tests, the test information is stored in a RRD database[1]. From this data averages are calculated and eventually the graphs are generated[10]. The results of a performance test will be compared with the results of previous tests and if the performance result of a test drops below a certain threshold, an alarm is raised. When an alarm is raised, an email is sent to the last contributor of the code of which a test fails. The same happens when one or more functional tests are failing.

The second part of the Codmon framework are the sensor files. A sensor file describes a "test set" that can be executed by the Codmon program. Such a sensor file consists of two parts. First there is the *onoff* part, which is used for compilation parts and the functional tests. Second there is the *graph* part. This part is used for the performance tests. The core of each sensor element in a sensor file is the CMD attribute. This CMD attribute consists of two basic parts: a wrapper and a shell-script command. This shell-script command can be anything. For instance it can be a SVN command, or a ANT job or something

[1] At this moment both the das2 and Codmon aren't in use anymore.

else. Listing 1 gives an example of such a senor.

```
<sensor id="update_ok"
    name="CVS update"
    cmd="perl CODMON_HOME/codmon/wrappers/time_wrapper.
        plCODMON_HOME sh codmon/local/cvsup.sh"
    scope="ibis"
    scheduled="false"
    graph="true"
    fatal="true" />
```

Listing 1: *Sensor example*

In this case the CMD attribute apparently consists of a wrapper called *time_wrapper.pl* and a shell-script command called *cvsup.sh*. The wrapper indicates the kind of test that is executed. The shell-script command indicates which program is tested. In case of Listing 1 the update time of the CVS repository is is measured.

Where a *sensor* describes the structure of a set of tests, a so called *wrapper* describes the actual test. Wrappers are small programs that are written int the PERL language. The return value of a wrapper indicates if a test was successful or not. In case of a failure an alarm is raised and both the return value and the actual error code will be mailed to the programmer who made the last contribution to the code. The results of the performance tests are plotted in a graph, which makes it easy for the developers to see the performance behavior. It is fundamental to understand that a *wrapper* is not part of the Codmon program itself. Leaving the wrappers outside of Codmon gives a user the possibility to add their own wrappers as well as using general wrappers without knowing the the details of the actual Codmon program. A general wrapper is a wrapper that can be used for each software component. The *time_wrapper* is a good example of a general wrapper. Figure 1 shows a schematic picture of the Codmon structure [10]. More technical details about the Codmon implementation can be found in [10].

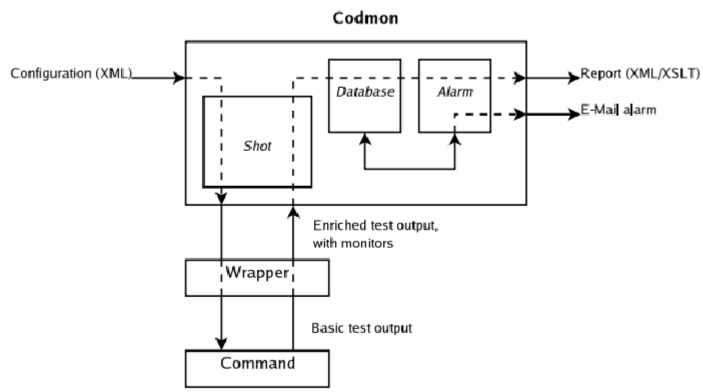
2.2 Codmon problems

The goal of this research was to see if it's possible to design a multi-platform, user friendly modular test environment. There are multiple reasons why Codmon doesn't fit the bill. In this section we'll discuss these reasons in detail.

2.2.1 Multi-platform

First Codmon is not multi-platform. To be multi-platform, without applying every change multiple times, the platform itself must be written in a platform-independent language. In Codmon at least three different languages are used. The Codmon program itself is written in Java, which is indeed platform independent[6], so this is not the real problem. As we've explained in section 2.1 the core-part of the sensors is a combination of a *shell-script* command and a *wrapper*. Since

Figure 1: *figure 1: Codmon*



a Linux shell-script usually won't work on a Windows environment this part is definitely not platform independent. The same can be said about the PERL language, this will without special effort, also not work on a Windows environment. Next to this there are also several separate shell-scripts for example, for CVS-checkouts and the startup of the Codmon framework. Taking this into account, we can easily see that the Codmon framework is far from platform independent.

2.2.2 Modularity

The second one is the modularity of Codmon. Due to the chaos of different scripts and languages it is difficult for programmers to add new modules or tests to the Codmon framework. Adding new tests and modules should be straight forward

2.2.3 User-friendliness

The third reason is the user-friendliness of Codmon. Next to modularity, which also contributes to the user-friendliness of the test environment, Codmon requires quite a bit of configuration,of which you don't want to bother anyone, before a user is able to use it. In the reminder of this thesis we'll explain in detail how we have solved these and other issues in the Codmon-VM environment.

Another issue regarding to user-friendliness is the structure of the wrappers. As you could see in section 2.1 the CMD attribute of a sensor consists of two basic parts: a wrapper and a shell-script command, of which the shell-script command could be anything. From a users perspective it would be much easier if there was only one type of command possible, an Ant-job[2] for instance.

3 The Road to Codmon-VM

As we described in section 1.3 our goal is to develop a multi-platform, user friendly modular test environment. This environment must at least satisfy the following requirements. The most important requirement is that Codmon-VM is *multiple-platform*. A second requirement is that is should be *modular* and pluggable. This means that it is possible for both users to easily add new components to the environment. The same should apply to developers who maintain the environment. A third requirement is that Codmon-VM environment must be user friendly. We don't want to bother the users of the Codmon-VM environment with the internal mechanisms of it. A second issue regarding to user-friendliness is that the sensors as described in section 2.1 should all have the same structure so a user can add tests always in the same way.

3.1 General decisions

In the previous sections we have stated that Codmon has some good aspects as well as some aspects that aren't that good. To design a test environment which satisfies the mentioned requirements we decided to reuse the good parts of Codmon and design new components where necessary. Since the core program of Codmon is written in Java and Java is multi-platform we decided to reuse most of this code and only adapt it if there is no other option. In section 4 we'll show where, how and why we've adapted the Codmon core program.

3.2 Multi-platform

Because the Codmon core program is written in Java already we've decided to write the all wrappers in Java as well. Using as few languages as possible will help the people who have to maintain the environment. We also saw that the core of a sensor consists of two parts a wrapper and a shell-script command. In most cases this shell script command will start some program. This program should also be able to run on multiple platforms. As we discussed above in Codmon-VM all the wrappers will be written in Java. Hereby writing these programs in Java as well is an obvious choice. So now we have a Java wrapper and a Java program. We thus need a platform independent construction that, at least, is able to start a program. We have chosen to use *Ant* for this. One of the reasons to choose Ant is that it provides us with a lot of flexibility. Lets take a look at the following example. A user who wants to test a piece of Java software probably already has a build file for building the software. So there is nothing new here. The only thing the user has to add is a new ant-target, which starts the program. Listing 2 shows the new sensor structure:

```
<sensor id="update_ok"
    name="CVS update"
    cmd="java <wrapper> <path to build file > <ant> <target>"
    scope="ibis"
```

```
    scheduled="false"
    graph="true"
    fatal="true" />
```

Listing 2: *Sensor new structure*

The <target> is an optional value. When the target value is left out, Codmon will run the default ant target. In section 4 we'll show how this all is implemented in the Codmon-VM environment.

3.3 Modularity

Another problem we discussed in section 2.1 is the modularity of the environment. The way Codmon was designed it could only handle CVS-checkouts and updates. The Codmon-VM environment must be able to deal with different versioning systems. To achieve this we decided to design separate *modules* outside of the Codmon-VM core program that takes care of the checkouts and updates of the software that the users want to test. Each of these modules implement the interface for a different version-control system. Adding support for a new version-control system implies that the codmon developers (or the users of Codmon-VM) have to write a new module that implements this interface for this new version-control system. This interface should at least define mechanisms for fetching and updating the software. Next to this is must also provide some basic mechanism which provide the users with the history log of the software. The only thing a user has to do, is indicate which version control system his software is using. We discuss the implementation in more depth in section 4.

The new sensor structure as discussed in section 3.2 makes adding new test also straight forward. The only thing a Codmon-VM user has to do is add a new test, and create an Ant target for this test.

3.4 User-friendliness

When a user wants to test his software on multiple platforms you don't want to download, install and configure the test environment over and over. The Codmon-VM project comes up with a nice and powerful solution for this problem. It provides a set of virtual machines in which Codmon-VM is already installed en pre-configured. The only things a user has to do before he can use the by Codmon-VM supplied tests is to download one or more of these virtual machines, load them into VM-ware and add the details of his software to an initialization file. By doing it this way the nasty configuration details are hidden for the users of the Codmon-VM environment. Also when a a new version of an operating system arrives, the Codmon-VM developers only have to configure one new image, which is directly available for all the users. How this works exactly we'll see in section 4.

4 The implementation of Codmon-VM

In This section we'll describe how we've implemented the decisions made in section 3 in the Codmon-VM project. We'll start with the parts that were necessary to achieve that Codmon-VM is multi-platform followed by the parts that affect the modularity. We end this section with the parts that affect user-friendliness. When someone wants to use the Codmon-VM test environment there are two options he can choose to get access to this environment. We'll come to these options later in this section. For now we assume the user has access to a working Codmon-VM environment.

4.1 multi-platform

In the previous sections we mentioned that the tangle of shell-scripts and PERL-code ensures that Codmon isn't multiplatform. To ensure that Codmon-VM is multiplatform we've chosen that all the code of the Codmon-VM environment is written in Java. First we describe how we've implemented the start-up of a test run. After this we describe how both the tests and the software that is tested are connected together.

4.1.1 Start-up

In a non multi-platform environment it is very easy to create a shell-script (or a batch-file in case of Windows) to start a program. As we wil see in Java this is a little more complex.

As we will see in section 4.2 everything outside the Codmon-VM core program is a separated module, so also the start-up module. The start-up module is called with one parameter namely the name of the sensor file that is needed for this run. The current start-up module is responsible for a few things. First it initializes the Codmon-VM environment. If it the first time Codmon-VM is used, four result directories are created. Otherwise the results of the previous test-runs are copied to the history folders so in the current result folder is space for the new test results [2].

When the all the result files are in the right place the Codmon-VM core program must be started. Because this is a completely new Java program we needed a way to pass the sensor file parameter to the core program. Remember at this stage Codmon-VM is already built. So to achieve this we have to make use of *dynamic class loading*. Dynamic class loading means that only at runtime is decided which (Java) class is called. Listings 7 and 8 in Appendix A show how this is implemented in the Codmon-VM environment.

[2]Due to historic reason the reusult folders are still called dday,dday1,dda2 etc etc

4.1.2 Ant-connector

A second issue we had to deal with were the the senors in the senor files. In section 3.2 we have already seen the new sensor structure. A wrapper is a separated process which has to be started by the Codmon-VM core program. When the Codmon-VM core program evaluates a sensor it first extracts the the wrapper from the sensor *cmd*-attribute. After this also the Ant-target is extracted from the *cmd*-attribute. If we use the senor of listing 3 the wrapper will be the *TimeWrapper*, while the Ant-target will be *"run"*.

```
<sensor id="checkout_ok" name="TestApps: checkout projects"
  cmd="java CODMON_HOME/codmon/wrappers/classes/TimeWrapper ../../local/checkoutApplications ant run"
  scope="checkOut"
  scheduled="false"
  graph="true"
 fatal="true"/>
```

Listing 3: *sensor example*

When the Codmon-VM core program has extracted these values it starts a *Wrapper* process (See appendix C). Listing 4 shows how this is done. The Ant target is passed as a parameter to this Wrapper process. This Wrapper,which is a test, starts he Ant-conector, which is a small Java Class (see Appendix C) which evaluates and executes the Ant-target. Such an Ant-target can be a "simple" *build* target or in this case it is the *run* target. This is shown in listing 5. This "run"-target executes the software that is tested by, in this case, the TimeWrapper.

```
final Process pr = new ProcessBuilder(argList)
                .directory(new File(dir))
                .start();
```

Listing 4: *Create new process*

```
        <!--Run the Checkout programm-->
        <target name="run">
                <java  fork="true" classname="Checkout" classpath="${env.CLASSPATH}" output="out.txt">
                        <classpath>
                                <path location="${jar.dir}/CheckoutApplications.jar"/>
                        </classpath>
                        <arg value="${arg0}" />
                        <arg value="${arg1}" />
                </java>
        </target>
```

Listing 5: *The run-target*

By implementing it this way it doesn't matter which (Java-) software the wrapper wants to test, it always be done in the same way as described above.

4.2 Modularity

In section 4.1 we described the improvements we made to achieve that Codmon-VM is multi-platform. In this section we'll describe how easy it is to add new modules to the Codmon-VM environment. To do this we use a module called *CheckoutApplications*.

4.2.1 CheckoutApplications module

Before Codmon-VM is able to test any software, this software must be available for the Codmon-VM environment. To be available this software must have been checked-out[3] from the repository where it is stored. The CheckoutApplications-module is responsible for this job. The CheckoutApplications itself consists of a main checkout module and different sub-modules which support a specific version control system. At this moment the CheckoutApplications module supports two different version control systems, namely Subversion (*SVN*) and Git.

The main checkout module is responsible for a few things. First it reads the initialization file, which contains information about the test projects. We'll come back to the initialization file later. This information contains, among other information, the kind of version control system that is used for this software. With this information both the update and updateLog methods for this software can be invoked (See Appendix D). For implementing this menthods for SVN repositories we've used the *SVNKit* Api[16]. For the Git repositories we've used the *JGit* Api[8].

As you can see now it is easy to add a sub-module for a new kind of version control system. The only thing a developer has to do is implement the update and updateLog methods. And add the calls to this methods to the *checkoutProject* method which you could see in Appendix D.

4.2.2 Tests

Adding new tests is relatively straight forward. As long as the test is written in Java a user can just add a reference to this test to the CMD-attribute of a sensor.

4.3 user-friendliness

Before the Codmon-VM environment is able to run one or more tests on a software, it has to know where it can find the software and check it out.This information and other information has to be added to an *init.xml* file. This file is stored at *"codmon/codmon/local/checkoutApplications/"*. Listing 6 shows the structure of the init file. The explanation of its elements is shown in table 1.

[3]for the sake of simplicity,when we talk about it in a general way, we call both check-out and clone check-out in this thesis.

projects	List of all the projects that are involved in a test series.
project	Contains the information of one specific project.
name	The name of the project.
location	The URL of the location where the project can be found.
versionControl	Element that contains specific information about the version-control system
type	The type of version control system that is used for the project. e.g. *SVN* or *Git*
run	**Boolean** that indicates is a project should be tested.
user	username to login into the version-control system
pwd	password to login into the version-control system

Table 1: Table1 : init.xml elements

```
 <projects>
        <project>
                <name>projectname</name>
                <location>http://www.sampleurl.com</location>
                <versionControl>
                        <type>svn</type>
                        <command>checkout</command>
                </versionControl>
                <run>true</run>
                <user>username</user>
                <pwd>pwd</pwd>
        </project>
<projects>
```

Listing 6: *init.xml example*

5 evaluation

TODO: This section evaluates the choices that are made in the previous sections'

6 Conclusion and related work

TODO: give answers to the questions from section Problem statement TODO: Discus related and future work

References

[1] ”JRobin 1.4.0.”. http://www.opennms.org/wiki/jrobin.

[2] ”Apache Ant”. http://ant.apache.org/.

[3] ”Das-2”. http://www.cs.vu.nl/das2/.

[4] ”Hudson Documentation”. http://www.hudson-ci.org/docs/.

[5] ”Jenkins Documentation”. http://jenkins-ci.org/.

[6] ”Henry McGilton” ”James Gosling”. The java language environment: Contents. May 1996. Section 4.2.

[7] Henri E. Bal Jason Maassen, Thilo Kielmann. GMI: Flexible and efficient group method invocation for parallel programming. March 2002.

[8] ”JGit”. http://download.eclipse.org/jgit/docs/latest/apidocs/.

[9] ”Junit”. http://junit.org/.

[10] ”François Lesuer”. Codmon: a source code monitoring tool. August 2005.

[11] Thilo Kielmann Markus Bornemann, Rob V. van Nieuwpoort. MPJ/Ibis: a flexible and efficient message passing platform for Java. pages 217–224, September 2005.

[12] ”Nunit”. http://nunit.org/.

[13] ”Winston Prakash”. Introducing hudson.

[14] ”The Ibis Project”. http://www.cs.vu.nl/ibis/.

[15] Thilo Kielmann. Henri E. Bal Rob V. van Nieuwpoort, Jason Maassen. Satin: Simple and efficient Java-based grid programming. *Scalable Computing: Practice and Experience*, 6(3):19–32, September 2005.

[16] ”SVNkit”. http://www.svnkit.com/.

[17] ”Masato Shinagawa” ”Toshiaki Kurkowa”. Technical trends and challenges of software testing. *Science and technology trends*, 29, 2008.

[18] Rob V. van Nieuwpoort, Jason Maassen, Gosia Wrzesinska, Rutger Hofman, Ceriel Jacobs, Thilo Kielmann, and Henri E. Bal. Ibis: a flexible and efficient Java based grid programming environment. *Concurrency and Computation: Practice and Experience*, 17(7-8):1079–1107, June 2005.

A Dynamic class loading

```
private ClassLoader getClassLoader(String[] jars) throws
    MalformedURLException, SecurityException{
                ArrayList<URL> paths = new ArrayList<URL>();

                for (String externalJar : jars) {
                        paths.add(new File(externalJar).toURI().
                            toURL());
                        System.out.println(paths.get(0));
                }

                URL[] urls = paths.toArray(new URL[paths.size()]);
                return new URLClassLoader(urls);
        }

        /**
         *@author bvl300
         *Loads codmon.jar so I can Use it here
         **/
        private Method getStartMethod(String[] argv){
                Method m = null;
                Class<?> cl = null;
                String[] jars = getJars();;
                try{
                        ClassLoader loader = getClassLoader(jars);
                        cl = loader.loadClass("Stats");
                        m = cl.getMethod("main", new Class[] { argv
                            .getClass() });
                }catch(Exception e){
                        System.out.println(e.getMessage());
                }

                if(!m.isAccessible()){
                        final Method temporary_method = m;
                        AccessController.doPrivileged(new
                            PrivilegedAction<Object>() {
                                public Object run() {
                                        temporary_method.
                                            setAccessible(true);
                                        return null;
                                }
                        });
                }
                return m;
        }
```

Listing 7: *dynamic class loading*

```java
/**
*@author bvl300
*Invoke method m with the correct parameters
*/
private void run(Method m,String[] argv){
        String sensor = argv[0];
        String[] statsArgs = new String[2];
        statsArgs[0] = "../sensors-"+sensor+".xml";
        statsArgs[1] = "../dday/shot-"+sensor+".xml";
        try{
                m.invoke(null,new Object[]{statsArgs});
        }catch(Exception e){
                System.out.println(e.getMessage());
        }
}
```

Listing 8: *invokation of the method*

B The TimeWrapper

```java
import java.text.DecimalFormat;
/**
 *@author bvl300
 *This wrapper messures the time of the module
 *that is executed.
 */
public class TimeWrapper{


        public TimeWrapper(String argv[]){
                String dir = argv[0];
                String cmd = argv[1];
                String target;
                if(argv.length==3){
                        target = argv[2];
                }else{
                        target = "main";
                }
                long startTime;
                double duration;

                Ant ant = new Ant(dir,target);
                ant.init();

                startTime = System.nanoTime();
                try{
                        ant.run();
                }catch(Exception e){
                        System.out.println(e+"\n<br/>\n");
                }finally{
                        duration = (double)((System.nanoTime()-startTime)/1000000000.0);
                        DecimalFormat df = new DecimalFormat("#.##");
                        System.out.println("<test id=\"time\" name=\"Time\" value=\""+df.format(duration)+"\" unit=\"s\"/>\n");
                }
        }


        /**
        *@author bvl300
        * */
        public static void main(String argv[]){
                new TimeWrapper(argv);
        }

}
```

Listing 9: *The TimeWrapper*

C The Ant-connector

```java
import java.io.File;
import org.apache.tools.ant.ProjectHelper;
import org.apache.tools.ant.Project;
import org.apache.tools.ant.ProjectHelper;

public class Ant{
        File buildFile;
        Project project;
        ProjectHelper projectHelper;
        String dir;
        String target;

        public Ant(String dir,String target){
                this.dir= dir;
                this.target = target;
                buildFile = new File(dir+"/build.xml");
                project = new Project();
                projectHelper = ProjectHelper.getProjectHelper();
        }

        public void init(){;
                project.setUserProperty("antFile",buildFile.
                    getAbsolutePath());
                project.init();
                project.addReference("ant.projectHelper",
                    projectHelper);
                projectHelper.parse(project,buildFile);
        }

        public void run(){
                this.project.executeTarget(target);
        }

}
```

Listing 10: *The Ant Class*

D Checkout Applications

```java
private void checkoutProject(Node project) throws SVNException, NoFilepatternException, GitAPIException, WrongRepositoryStateException, InvalidConfigurationException, DetachedHeadException, InvalidRemoteException, CanceledException, RefNotFoundException, NoHeadException, IOException {
        String url, type, projectName, user, pwd, command;
        long rev= -1;
        if (project.getNodeType() == Node.ELEMENT_NODE) {
                        Element eElement = (Element) project;

                url = eElement.getElementsByTagName("location").item(0).getTextContent();
                type = eElement.getElementsByTagName("type").item(0).getTextContent();
                command = eElement.getElementsByTagName("command").item(0).getTextContent();
                projectName = eElement.getElementsByTagName("name").item(0).getTextContent();
                user = eElement.getElementsByTagName("user").item(0).getTextContent();
                pwd = eElement.getElementsByTagName("pwd").item(0).getTextContent();
                if(type.equals("svn")){
                        SVN svnRep = new SVN(basePath, projectName, user, pwd, url, command);
                        if("checkout".equals(command)||"export".equals(command)){
                                svnRep.update();
                        }
                        rev = svnRep.getRev();
                        if(checkOldLog(projectName, rev)){
                                svnRep.updateLog();
                        }
                }else if(type.equals("git")){
                        GitObject gitRep = new GitObject(basePath, projectName, url, user, pwd);
                        if("clone".equals(command)){
                                gitRep.update();
                        }
                        if(checkOldLog(projectName, rev)){
                                gitRep.updateLog();
                        }
                }else{
                        //throw SVNException("Version control system not found");
                }
        }
}
```

Listing 11: *Invoking the right update method*