Codmon: A multi-platform modular test environment.

Berend van Veenendaal

March 23, 2014

TODO:Abstract

Preface

TODO: Preface,acknowledgements

Contents

1 Introduction

This chapter will give an introduction about the Codmon 2.0 project by giving a brief decription 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 [13]. One of the challenges of software engineering is to make sure that the software behaves in the same way on different platforms. Even when software is written in such a way that it can run on multiple platforms, there are still issues that must be dealt with, before one is able to run and test the software. Example issues are the configuration of the test environment or finding and installing all the prerequisite libraries etc etc.

1.2 Problem indication

Nowdays there are numerous test frameworks and test environments available. For example there is *Junit*[6] for Java-unit testing and *NUnit*[9] for C#-unit testing. There are also different environments like Hudson[2], [10], Jenkins[3] which can build a project and run a series of (unit) tests against this project. All of the frameworks and environments have both their advantages and disadvantages. One of the advantages of unit testing is that a software developer easily can 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. One of them is, although they both run on different 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 to be able to do this, a user must know in advance on which platform this script has to run. With this in mind you can see that, 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 2.0. The Codmon 2.0 project provides users with a set of virtual machines, in which Codmon 2.0 is already installed and preconfigured. The purpose of the virtual machines is to make it as user friendly as possible. By doing it this way the only tasks a Codmon 2.0 user has to do are 1) add their project to the init.xml file. and 2) add the tests to a so called wrapper file. Both of these files will be discussed in more detail in sections 3.1.1 and 3.1.3 .

1.4 Thesis outline

Section 2 first describes the current Codmon framework. It starts with explaining why the original Codmon was built. Then it continuous with a description of the design of Codmon and the problems of the current Codmon project in section 2. Section 3 describes the Codmon 2.0 project. This section starts with the *The road to Codmon 2.0*. Here we explain the ideas that came into mind and how we got to the final Codmon 2.0 design. Section 3 describes the Codmon 2.0 project. It starts with a general description of the project followed by a detailed explanation of the different modules of Codmon 2.0. In Section 4 we discuss the results based on section 3.2. We end this section with a brief discussion of related work.

2 Codmon

Before we start describing the Codmon 2.0 we start with a description of the current Codmon framework, we first describe How Codmon works and why it is not good enough for the purposes mentioned in section 1.3. The original Codmon framework was built in 2005 by François Lesuer[7]. Codmon is built for testing Ibis projects[11][12][8][14][5] on the DAS-2[1] computer. Codmon was (At this moment the das2 and Codmon aren't in use anymore) able to perform both functional and performance tests. If for some reason a particular test failed, Codmon reported the problems directly to the programmer who made the last changes in code under test. It was also the intention that Codmon would be extensible. In this the Codmon developers succeeded only partially. We will discuss this more in depth in section 2.2.

2.1 Codmon Design

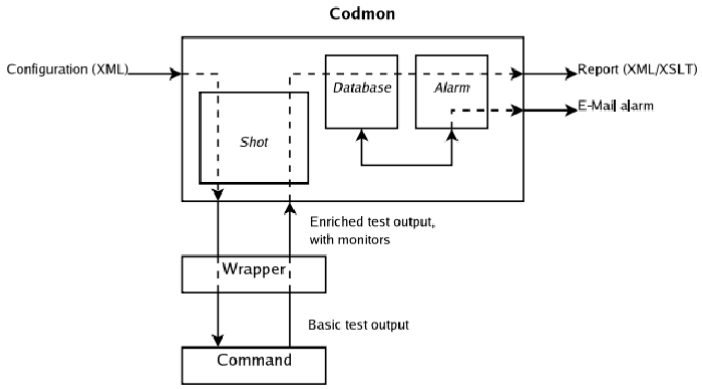
This section we will describe the design of the Codmon framework. The setup of Codmon is more or less modular and consiste of a few core parts. The set of tests that should run is describe in different configuration files which are called *Sensors*. A sensor describes which test should be executed and also the software that is under these tests. A sensor consits of two parts. First there is the *onoff* part, which is used for compilation parts and functional tests. The core of each sensor element in a sensor file consists of two parts: a wrapper and a shell-script command. Second there is the *graph* part. This part is used for performance tests. 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 level, an alarm is raised. The same occures when one or more functional tests are failing.

Where a *sensor* describes the structure of a set of tests, a so called *wrapper* describes an actual test. Wrappers are small programms that are written int the PERL language. The return value of a wrapper indicates if a test succeeded or not. In case of a failure an alarm is raised and both the return value and the actual error code will be shown 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 behaviour. Figure 1 shows a schematic picture of the Codmon structure. More technical details about the Codmon implementation can be found in [7]

2.2 Codmon problems

The purpose 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 satisfy this. First to be multi-platform, without applying every change multiple times, the platform must be written in a platform-independent language. If we take a look ad Codmon there are at least three different lang used. The core of Codmon is written in java, which is indeed platform independent[4], so this is not the real problem. As

Figure 1: *figure 1: Codmon*



we've explained in section 2 the important part of the sensors is a combination of a *shel-script* command and a *wrapper*

TODO: Was only supporting cvs. difficult to extend for other users. TODO: Think about too many different languages and techniques (java , perl) the later isn't platform independent.

3 Codmon 2.0

3.1 the road to Codmon 2.0

//TODO: Think of better subsection title!! //TODO: Explain ideas and road to solution

3.1.1 The init.xml file

3.1.2 Version control

3.1.3 wrappers

3.2 experiments

4 Conclusion and related work

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

References

[1] "Das-2". http://www.cs.vu.nl/das2/.

[2] "Hudson Documentation". http://www.hudson-ci.org/docs/.

[3] "Jenkins Documentation". http://jenkins-ci.org/.

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

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

[6] "Junit". http://junit.org/.

[7] "François Lesuer". Codmon: a source code monitoring tool. August 2005.

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

[9] "Nunit". http://nunit.org/.

[10] "Winston Prakash". Introducing hudson.

[11] "The Ibis Project". http://www.cs.vu.nl/ibis/.

[12] 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.

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

[14] 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.