Codmon: A multi-platform modular test environment.

Berend van Veenendaal

March 25, 2014

TODO:Abstract

Preface

TODO: Preface,acknowledgements

Contents

1 Introduction

This chapter introduces the Codmon 2.0 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 [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. 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 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. 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 2.0*, we explain the ideas that came into mind and how we got to the final Codmon 2.0 design. Section 4 describes the implementation of 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. After this we will evaluate the choices and the consequences of these choices 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 shortcommings. The original Codmon framework was built in 2005 by François Lesuer[7]. Originally Codmon was built for testing and performance monitoring 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. It does this by sending an email directly to the programmer who made the last changes in code under test. Codmon also reported in the same way in case the performance drops below a certain treshold. The 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 Codmon Design

In this section we will describe the design of the Codmon framework. The setup of Codmon is more or less modular and consists of a few core elements. The set of tests that should run is described in different configuration files which are called *Sensors*. A sensor describes precisely which tests should be executed and also the software that is under these tests. A sensor consists of two parts. First there is the *onoff* part, which is used for compilation parts and the 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 the 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 treshold, 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.

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. 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 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. First, to be multi-platform, without applying every change multiple times, the platform itself must be written in a platform-independent language. If we take a

Figure 1: *figure 1: Codmon*
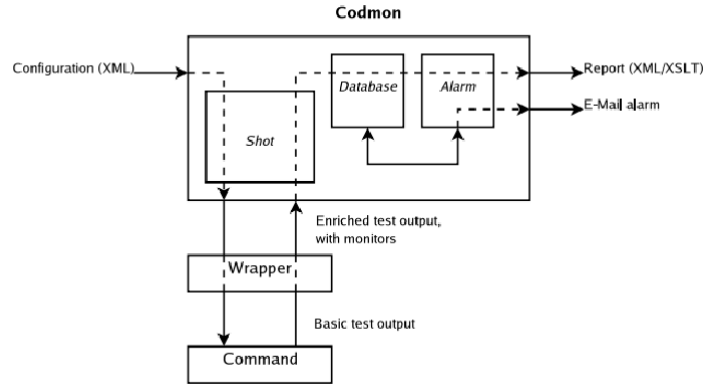
look at Codmon there are at least three different languages used. The core of Codmon is written in Java, which is indeed platform independent[4], so this is not the real problem. As we've explained in section 2.1 the core-part of the sensors is a combination of a *shell-script* command and a *wrapper*. Since a Linux shell-script will 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 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. Due to the chaos of different scripts and languages it is also difficult for programmers to add new modules or tests to the Codmon framework. In the reminder of this thesis we'll explain how Codmon 2.0 solves these and other issues.

3 The Road to Codmon 2.0

//TODO: Explain ideas and road to solution

4 The implementation of Codmon 2.0

//TODO: Section describes the implementation of the Codmon 2.0 project.

4.1 The init.XML file

4.2 Version control

4.3 wrappers

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 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.