Development and Operations[*]

Christoffer Holmstedt
christoffer.holmstedt@gmail.com

ABSTRACT

This report includes lessons learned from the Naiad project about ”development and operations” (DevOps) such as the version control system used as well as how our website was set up. The report also touches on the IT infrastructure set up in the project room for development and testing purposes.

1. INTRODUCTION

No matter what you're developing a devlopment process has to be set up and followed. As project teams grow bigger and bigger, the development process gets more important. For the Naiad project we were 18 students and most of us did some development. In close relation to the development comes operations, which includes the installation and management of the Naiad website (Wordpress, MySQL and backup procedures).

This report will start of with lessons learned from the development process and continue on with lessons learned from operations. The conclusion section in the end will list the most important improvements that can be made in the next year project.

2. LESSONS LEARNED

2.1 Development

The requirements we got from our customers included one specific on the programming language to use, Ada. In the beginning this felt like a bad choice as the members of the project team had limited experience with Ada but we couldn't do anything about it. Instead we took height for a longer learning period and a more open mindset for future trouble such as which compilers would work and for which architectures the Ada features we needed could be used on.

[*]This report was written during the fall of 2013 in an advanced level project course at Mälardalen University, Sweden.

2.1.1 Ada

Ada is a strong-typed language. The first release of the language was published in 1983 with updates in 1995, 2005 and the latest in 2012. With the limited experience of Ada within the project team we didn't decide upon a specific version to use early on. It later turned out that it would have been wise to limit us to Ada 2005 functionality for better portability between different versions of the compiler used.

Compilation and **cross-compilation** was done with the ”GNU Compiler Collection” (GCC) compiler including the GNAT frontend. When someone is talking about ”gnat” they are basically talking about GCC compiled with Ada programming language support. As of January 2014 Ada support is not compiled by default with GCC, though packages in Debian/Ubuntu exists for easy installation.

In the beginning of the project we were recommended to use the GNAT GPL compiler supplied to academia for free by AdaCore. The latest release was from April 2013. The GNAT GPL compiler from AdaCore is basically the GCC compiler with some extra features and bundled packages from AdaCore. The reason for the recommendation was that the GNAT GPL version earlier included a compiler for AVR architecture as well, this support was dropped with the 2013 release from AdaCore. There were discussions made about aquiring a GNAT Pro version from AdaCore to be able to compare functionality between the GPL and Pro version but the Pro version was never aquired.

As development platforms Windows 7/8 and Ubuntu 12.04 LTS/13.10 were used. Instead of trying to aquire the GNAT Pro license we looked into the available packages in the Ubuntu repository (those with Windows still used the AdaCore GNAT GPL version). The GNAT packages available in Ubuntu is directly imported from the Debian project and have a strict policy [6] how they should be compiled so every Ada package available in the repository should work with all others. The downside of this policy is that most packages are quite old but the upside is that they all work together and are easy to install.

For cross-compilation against the AVR platform the AVR-Ada project [3] is available. It requires GCC 4.7.X and the supplied package in Ubuntu is gnat-4.6 so some GCC compilation is required to be done to get it to work on Ubuntu.

We also used the BeagleBone Black in the project which is an ARM processor with the ARMv7 architecture. As we installed Ubuntu directly on the BeagleBone Blacks we could also install the Ubuntu package and compile Ada code natively directly on the BeagleBone Blacks, we never tried to cross-compile for ARM.

In the end on Windows we use the AdaCore GNAT GPL bundle from 2012 to be able to compile for the AT90CAN128 micro controller (AVR architecture). Compilation for the BeagleBone Blacks was done with gnat-4.6 available in the repository, natively on the board. As a lot of development and testing took place on Ubuntu 64-bits machines both AdaCore GNAT GPL bundle from 2013 and the gnat-4.6 package from the repository were used. One important aspect to remember here is that if the GPS IDE is used remember to set the compiler to 2005 version otherwise some features might compile on one machine and not on others. We noticed this when first trying out our code on a development machine with the amd64 architecture and then compiling the same code on the BeagleBone Blacks, it didn't work.

As final pointers it's worth mentioning that compiling for the AVR architecture on Ubuntu is very limited, it works but support for specific MCUs is limited. During Naiad project some development was made to support the AT90CAN128 but the work was never finished. Also, compilation for the AVR architecture doesn't support any object-oriented programming such as tagged types and interfaces nor Ada tasks, this is true for both the AdaCore GNAT GPL bundle and the gnat-4.6 package from Ubuntu repository.

Online help is available but very limited when it comes to Ada. Great channels for Ada help are the newsgroup comp.lang.ada [14], the #Ada IRC channel on the Freenode network as well as related mailing lists [4, 5].

As recommended reading or at least reference litrature a few reports exist on Ada development. These are the "Ada Style Guide" [2], the "Guide for the use of the Ada programming language in high integrity systems" [10], the "Guide for the use of the Ada Ravenscar Profile in high integrity systems" [11], the "High-Integrity Object-Oriented Programming in Ada" reports [12] and the rationale behind Ada 2012 [1].

2.1.2 Version Control System

Available as a service from Mälardalens University is a restricted Subversion repository including code and other work from previous projects in the course. The repository size is pretty big so it's hard to work with. Early on in the project we decided to go with Git and Github as version control system. Though only few of the project members had used Git before so some training had to be done early on. This limitation also made us choose a centralised version control system workflow. Basically we set up a common repository on Github [9] and gave all project members access to it.

The use of Git and Github was a good choice, though in the end we have had some problems with the workflow e.g. that conflicts wasn't sorted out properly, especially on Windows machines where the TortoiseGIT GUI can be hard to understand from time to time. As suggestion for next year Git is the best choice for VCS though be prepared to spend some extra time to teach those that don't know Git from before about a proper distributed workflow such as git-flow [8] and put one project member as release manager which is the only one that has commit access to the main/master branch.

The release manager will act as a guard to the main branch so only working, tested and well written code is in the main branch. Another approach could be to look at setting up your own installation of "Gerrit" [7] as a web-based peer-reviewing tool. This would let everyone peer review everyone's code and you could limit merges to the main branch to patches with at least 2 approved peer reviews.

2.1.3 GPS with AUnit

For unit testing AUnit was used. AUnit comes bundled with the AdaCore GPL version and can be installed with the GNAT packages on Ubuntu. One main difference is that the GPS IDE version that is bundled with AdaCore GPL has more AUnit functionality than the version available in the Ubuntu repository.

2.1.4 Coding Guidelines

Early on in the project we sat down in the software team to decide upon coding guidelines. The goal with coding guidelines is that code should be easy to read and understand, primary by other project members so they can help you as fast as possible when you get stuck. Other project members should not have to spend hours trying to parse your code.

No matter what guidelines you chose someone has to enforce these otherwise project members will start to say that they will fix that later on, but that will never happen. Before deciding upon your own coding guidelines it's wise to look at the public "Ada Quality and Style Guide" available on Wikibooks [2].

2.1.5 Technical Reports in LaTeX

Just as keeping track of different versions of your Ada code, LaTeX text can be tracked with Git as well. One thing to remember is to use linebreaks when rows starts to get too long. The reason for this is that if a line consists of a complete paragraph and you change just a word in that paragraph Git will show that the entire paragraph has changed because it's written in a single line. If you instead break the paragraph with several linebreaks and then change a single word, only that line the tex file will be marked as changed.

2.2 Operations

"Operations" consists of all the software and hardware we had to maintain during the project. This includes some network equipment and a server machine the project room as well as the webserver installed on a virtual private server.

2.2.1 Continious Integration with AUnit

As mentioned earlier, AUnit was used for unit testing and the idea was to set up our server machine with some kind of continous integration testing server. After each commit to the Git repository the test server would run all tests and give a positive or negative response back if all tests passed or not. If any test failed the developer would be informed with details about which test didn't pass.

Jenkins CI [13] was set up though we didn't get any public IP address so we had to use a polling scheme for the test server so it checked for new code every 5 minutes. The lack of public IP also restricted us to only get access to the test results when we were at the university. The Jenkins service was up and running throughout the project but it wasn't used at all. In the end, the lesson learned was that if project members can't see the benefits of a service, it's not worth maintaining it.

2.2.2 *Website*

For the website it was decided that we wanted to use some kind of content management system and not develop it from scratch, time is better spent on other tasks. The experience within the project team suggested that we would go with Wordpress.

Wordpress was set up on a Ubuntu virtual machine at the hosting company Tilaa located in the Netherlands [16]. The price for the VPS was about 17 euro a month, including VAT. The rationale for this was that we could have used a service such as wordpress.com but end up with a lot of extra costs for changing the design as we want it. Wordpress.com is a good service when you don't want to change too much. Our own VPS gave us much more freedom in the design of the website.

For tracking visitors and the number of hits we used "Piwik" [15].

2.2.3 *Backup*

For backup of the website we used daily cronjobs on the server machine in the project room to pull down the current information available on the VPS.

3. CONCLUSION

A lot can be done with different tools to improve the efficiency of the development team and the quality of the code that is produced, though it's important to remember that too much of it will leave it all unused. Tools that are in the daily workflow are the ones that will be worth installing and maintaining.

3.1 Development

3.1.1 *Ada*

Try to stick with one development environment and make sure to set up cross-compilation of all your code straight away. This will prevent surprises were you end up rewriting a lot of code that works on the development machines but not on the ARM or AVR architectures.

To make sure all project members run with the same development environment you could look in to the possibility of setting a Vagrant [17] virtual machine system and letting everyone run with that.

3.1.2 *Version Control System*

Git with Github services was a good choice for version control system though the workflow chosen was not. The time spent learning a distributed workflow such as git-flow is time well spent early on in the project.

One last thing to remember is that a repository can grow very quickly if you allow binary files to be added. Some test files such as test images for the vision system are needed to be in the repository but try to limit it to as few as possible. This will keep the repository size to a minimum and make it easier to clone on new development machines.

3.2 Operations

3.2.1 *Website*

If possible try to use wordpress.com service or similiar. The burden of maintaining the website is not worth it and you will save some well needed cash as well. A public service such as wordpress.com will also make it easier to transfer the responsibility to others when the project course is over. The downside is that you won't have that much of flexibility when it comes to design.

4. REFERENCES

[1] Ada 2012 rationale. `http://www.adacore.com/knowledge/technical-papers/ada-2012-rationale/`. Accessed 2014-01-10.

[2] Ada quality and style guide. `https://en.wikibooks.org/wiki/Ada_Style_Guide`. Accessed 2014-01-10.

[3] Avr-ada compiler at sourceforge. `http://sourceforge.net/projects/avr-ada/`. Accessed 2014-01-08.

[4] Avr-ada devel mailinglist. `https://lists.sourceforge.net/lists/listinfo/avr-ada-devel`. Accessed 2014-01-08.

[5] Debian ada devel mailinglist. `https://lists.debian.org/debian-ada/`. Accessed 2014-01-08.

[6] Debian policy for ada. `http://people.debian.org/~lbrenta/debian-ada-policy.html`. Accessed 2014-01-08.

[7] Gerrit code review. `http://code.google.com/p/gerrit/`. Accessed 2014-01-10.

[8] Git-flow workflow. `http://nvie.com/posts/a-successful-git-branching-model/`. Accessed 2014-01-08.

[9] Github repository: naiad-auv-software. `https://github.com/naiad-auv/naiad-auv-software`. Accessed 2014-01-08.

[10] Guide for the use of the ada programming language in high integrity systems. `http://standards.iso.org/ittf/PubliclyAvailableStandards/c029575_ISO_IEC_TR_15942_2000%28E%29.zip`. Accessed 2014-01-10.

[11] Guide for the use of the ada ravenscar profile in high integrity systems. `http://standards.iso.org/ittf/PubliclyAvailableStandards/c038828_ISO_IEC_TR_24718_2005%28E%29.zip`. Accessed 2014-01-10.

[12] High-integrity object-oriented programming in ada. `http://www.adacore.com/knowledge/technical-papers/high-integrity-oop-in-ada/`. Accessed 2014-01-10.

[13] Jenkins ci. http://jenkins-ci.org/. Accessed 2014-01-10.

[14] Newsgroup comp.lang.ada. https://groups.google.com/forum/#!forum/comp.lang.ada. Accessed 2014-01-08.

[15] Piwik. http://piwik.org/. Accessed 2014-01-10.

[16] Tilaa - virtual private servers. https://www.tilaa.com/. Accessed 2014-01-10.

[17] Vagrant. http://www.vagrantup.com/. Accessed 2014-01-10.