Martin Ettl et al.: Continuous Software Integration and Quality Control during Software Development, IVS 2012 General Meeting Proceedings, p.227–230
http://ivscc.gsfc.nasa.gov/publications/gm2012/ettl1.pdf

Continuous Software Integration and Quality Control during Software Development

Martin Ettl [1], *Alexander Neidhardt* [1], *Walter Brisken* [2], *Reiner Dassing* [3]

[1] *Forschungseinrichtung Satellitengeodäsie (TUM), Geodetic Observatory Wettzell*
[2] *National Radio Astronomy Observatory*
[3] *Bundesamt für Kartographie und Geodäsie, Geodetic Observatory Wettzell*
Contact author: Martin Ettl, e-mail: ettl@fs.wettzell.de

Abstract

Modern software has to be stable, portable, fast, and reliable. This requires a sophisticated infrastructure supporting and providing the developers with additional information about the state and the quality of the project. That is why we have created a centralized software repository, where the whole code-base is managed and version controlled on a centralized server. Based on this, a hierarchical build system has been developed where each project and their sub-projects can be compiled by simply calling the top level Makefile. On the top of this, a nightly build system has been created where the top level Makefiles of each project are called every night. The results of the build including the compiler warnings are reported to the developers using generated HTML pages. In addition, all the source code is automatically checked using a static code analysis tool, called “cppcheck”. This tool produces warnings, similar to those of a compiler, but more pedantic. The reports of this analysis are translated to HTML and reported to the developers similar to the nightly builds. Armed with this information, the developers can discover issues in their projects at an early development stage. In combination it reduces the number of possible issues in our software to ensure quality of our projects at different development stages. These checks are also offered to the community. They are currently used within the DiFX software correlator project.

1. A Short Introduction to Continuous Integration

During the development of a software project the state and stability of the current development version is hard to determine. It is difficult to find side-effects between the modules, when there are updates in the source. Also portability issues, such as deprecated functions or 64bit-/32bit-compatibilities, are difficult to find, without dedicated checks. This situation becomes more complex when multiple developers work on resources, having relations to other projects. A first helpful step is to set up a centralized version control management system, to which each developer commits his changes regularly (at least once a day). All the different versions over the time line of software projects are managed and stored in this centralized software repository. Therefore all versions can be restored easily. This makes it very comfortable to revert to an older version of the source code trunk. Also the newest version of the source code is always available for the developers and also for automated inspections.

Static code checks, nightly builds on different platforms with multiple compiler versions, code beautifier runs, documentation generators, or even unit tests can be started automatically on the latest software version, to check the included source code (see Figure 1). The results of the inspections and tests are converted to HTML pages. These are published on project homepages [5] in password restricted areas, to which only the developers of the project can get access. Therefore, each developer can use this information to detect and fix possible problems in the code.

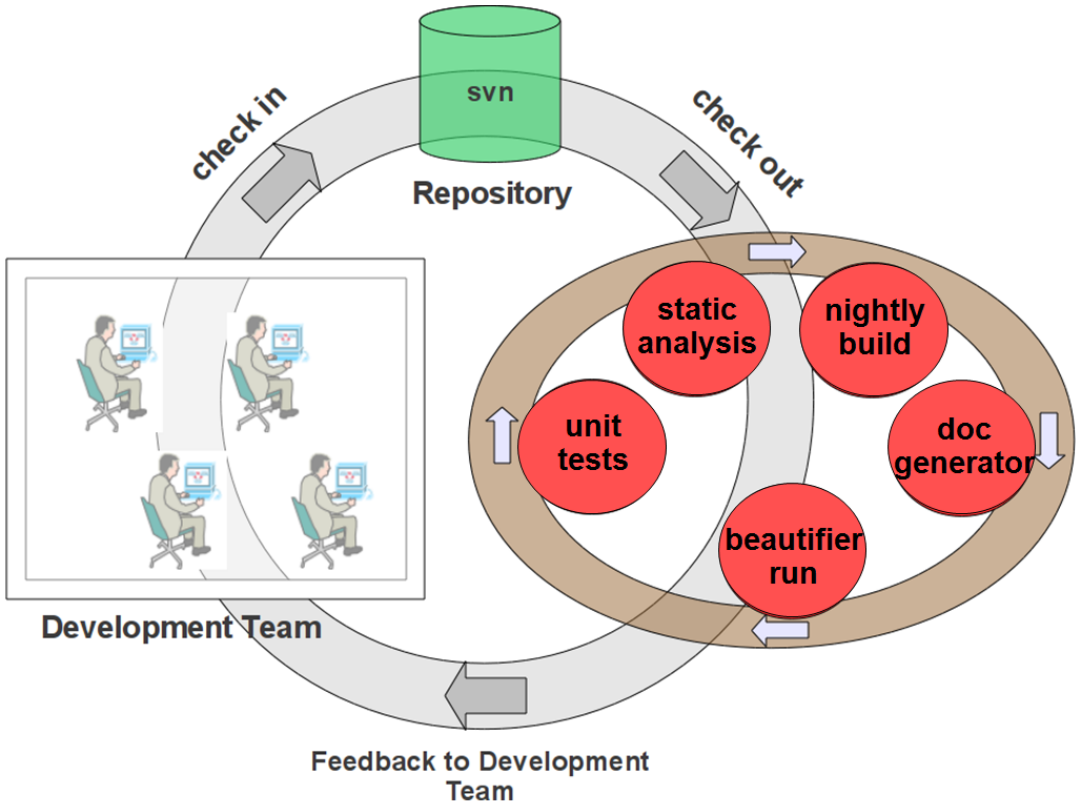

Figure 1. Software development including continuous integration mechanisms.

2. Different Methods of Code Inspection

In the continuous integration environment at the Geodetic Observatory Wettzell several different inspections are processed automatically each night. These are briefly introduced in the following sections.

2.1. Static Code Analysis

Static code analysis inspects the code to find potential programming flaws, without executing or compiling the source code. Currently a collection of open source static analyzing tools are used [3][8][6][7]. These tools aim to find bugs, which usually a compiler does not detect in C/C++-source code. It checks the code for memory leaks, null pointer dereferencing, unused variables, uninitialized variables, mismatching allocations/de-allocations, buffer overruns, out of bound memory accesses, and many others.

2.2. Nightly Builds

An automated build-system is based on standardized GNU-Makefiles for each project. Projects with several sub-projects have a top level Makefile, which is able to build all sub-projects at once. Therefore the whole code basis can be compiled by simply calling the top-level Makefile. Exactly this is done automatically on a Linux server every night, using several GNU-compiler versions. The build output is parsed, colorized, and converted to HTML for the presentation to the developers. Therefore they can check immediately if their committed source code has some effect on other builds in different configurations.

2.3. Documentation Generator

The developer documentation is created by Doxygen [4], an open source documentation generator. This tool reads the source code, including all the comments, and extracts the needed information to generate a developer documentation in different output formats with call-graphs and Unified Modeling Language diagrams. Running the documentation generation automatically supports a quick sharing of information. Furthermore, the generated documentation can be used to get an overview about the object-oriented software structure and the relationship of software components in the projects. This makes it easier for project beginners to understand the programming interfaces and the internal structures.

2.4. Spell Checker and Code Beautifier

Due to the limited spell checking capabilities of programming editors, an automated spell checking tool [2] helps to reduce the number of misspelled words in source code and ASCII files. Finally, this improves the readability of the code and reduces errors in the automatically generated documentation.

An automatic formatting tool [1] is used at regular intervals to format the source according to specific design rules. This ensures retention of the same indentation and text style in the whole software project and improves significantly the readability. In addition, it reduces maintenance time for developers and simplifies the sharing of source code.

2.5. Unit Tests

Unit tests are small test programs that check the plausibility of function behavior and results on the function level. A programming based environment to collect all the functional testing programs for different test cases is used (*simple_testsuite*). This suite validates all of the basic software components and the generated code. The suite runs on different architectures (32-/64-bit) with different compilers and in combination with different Linux operating systems to reveal portability issues. Furthermore, the test-coverage is measured using the GNU-compiler functionality. The information about the current test-coverage as well as the unit test reports give an overview of the current software test state. Based on this information, it is possible to measure the quality of the tested source code in a dedicated code metric.

3. Summary and Outlook

The continuous integration work-flow reduces the amount of severe issues during the whole software development phase at the Geodetic Observatory Wettzell. Currently, all software developments at the observatory are checked internally. Also the DiFX community uses this service on the e-Control Software Web Page [5]. An interactive usage of these possibilities helps to improve shared code to increase the quality factors.

Acknowledgements

The authors wish to thank especially the DiFX developer group for using the continuous integration Web environment.

References

[1] Artistic Style 2.02. A Free, Fast and Small Automatic Formatter for C, C++, C#, and Java Source Code. http://astyle.sourceforge.net/.

[2] codespell. Official codespell repository. http://git.profusion.mobi/cgit.cgi/lucas/codespell.

[3] cppcheck. Static analysis of C/C++ code. http://sourceforge.net/projects/cppcheck.

[4] Doxygen. Doxygen Manual. Generate documentation from source code. http://www.stack.nl/dimitri/doxygen/index.HTML.

[5] e-Control Software. http://econtrol-software.de/.

[6] Flawfinder. http://www.dwheeler.com/flawfinder/.

[7] nsiqcppstyle. C/C++ Coding Style Checker. http://code.google.com/p/nsiqcppstyle/.

[8] Splint. Annotation-Assisted Lightweight Static Checking. http://www.splint.org/.