Continuous integration and quality control during software development

Martin Ettl1,2, Reiner Dassing, Alexander Neidhardt1, Reiner Dassing
(FESG Wettzell, MPIfR Bonn, BKG)

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's 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, an 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 is called every night. The results of the build, along with the compiler warnings 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 is translated to html and similar to the nightly-build, reported to the developers. Armed with this information, the developers can reveal issues in their projects at an early development stage. This overall reduces the number of possible issues in our software to ensure quality of our projects at every development stage.

What does continuous integration mean?
During the development of a software project it is hard to determine the current state and stability of the current development version. Neither side-effects, nor portability issues can be detected in this development phase, especially when multiple developers working on resources affecting several projects behavior. The first attack to this problem was setting up a centralized version control management system where each developer commits his changes regularly to a centralized software repository. In this repository there are all the different versions stored and they can be restored easily. This makes it very comfortable, for instance to revert to an older version of the source code. Based on this, the newest version of the source code is always available and it can therefore tested intensively. The whole work flow of the continuous integration concept is depicted in the figure (right). First of all, the developer team is working on the project committing their changes to the software repository. By committing the changes frequently, at least once a day, the repository stores the latest version of the project. Based on the newest version of the software, every night a bunch of separated tests running against the source code (see red circles). Then, the results of the tests are converted to html pages. These results will be published in our local network. Therefore each developer can use this information to detect and fix possible issues of the latest changes on the code. This kind of work flow reduces the amount of severe issues during the whole development phase and helps the developers to find bugs in an early development stage.

What is static code analysis?
Static program analysis is done without executing or compiling the source code. For our analysis we are using cppcheck3, an open source analysis tool for C/C++-source code finding bugs a compiler does not detect. It checks our code for memory leaks, null pointer dereferencing, unused variables, not initialized variables, mismatching allocation-deallocation, buffer overrun, memory out of bounds checking and many more issues. The analysis report is converted to an html-page and published to the developers in order to be able to react on found problems. We consider to use in future other static analysis tools like flawfinder4 and sphinx5.

Why unit tests?
Unit tests are small test programs for checking the results at function or module level. For this, we have created an programming based environment for collecting all these small testing programs (simple testsuite). This suite offers a way to create such validation tests for all our basic software components and the generated code. This tests can be run on different architectures (32/64-Bit) with different compilers on different Linux operation systems to reveal portability issues. Furthermore the test-coverage is measured using the Intel compiler suite for Linux. The information about the test-coverage as well as the unit-test report is also provided to the developers. With this information it is now possible to measure the quality of the tested source code, which is very comfortable.

What is a code-beautifier?
We are using a tool called artistic-style1 which formats our source code automatically every week, according our design rules. This is done frequently and ensures the same syntax in our whole software. This therefore improves readability and reduces maintenance time for developers. Furthermore the takeover/sharing of source code between developers is simplified.

Why nightly builds?
We have created an automated build-system based on standardized gnu makefiles, where every project has its own makefile. Projects that contain several sub-projects have a top level makefile, capable of building all sub projects at once. Therefore the whole code base can be compiled by simply calling the top-level makefile. This is done automatically every night on our Linux-servers. Then all the compiler warnings will be piped to a text file and converted to a build report. This report containing possible errors and warnings is forwarded to our development team.

What is a documentation generator?
Our developer documentation is created by an open source documentation generator, called doxygen6. This tools reads the source code with all the comments, extracts the needed information and generates developer documentation in several formats along with call-graphs, Unified Modelling Language (UML)-diagrams. Running the documentation generation automatically, is a great help for our developers. This makes it simpler to share information between several developers. Furthermore, this generated documentation can be used to get an overview about the object-oriented software structure and the relationship of our software components in our projects. This for instance makes it easier for beginners to understand how a specific module operates and how it can be used.

Reference: