Continuous integration and quality control during software development

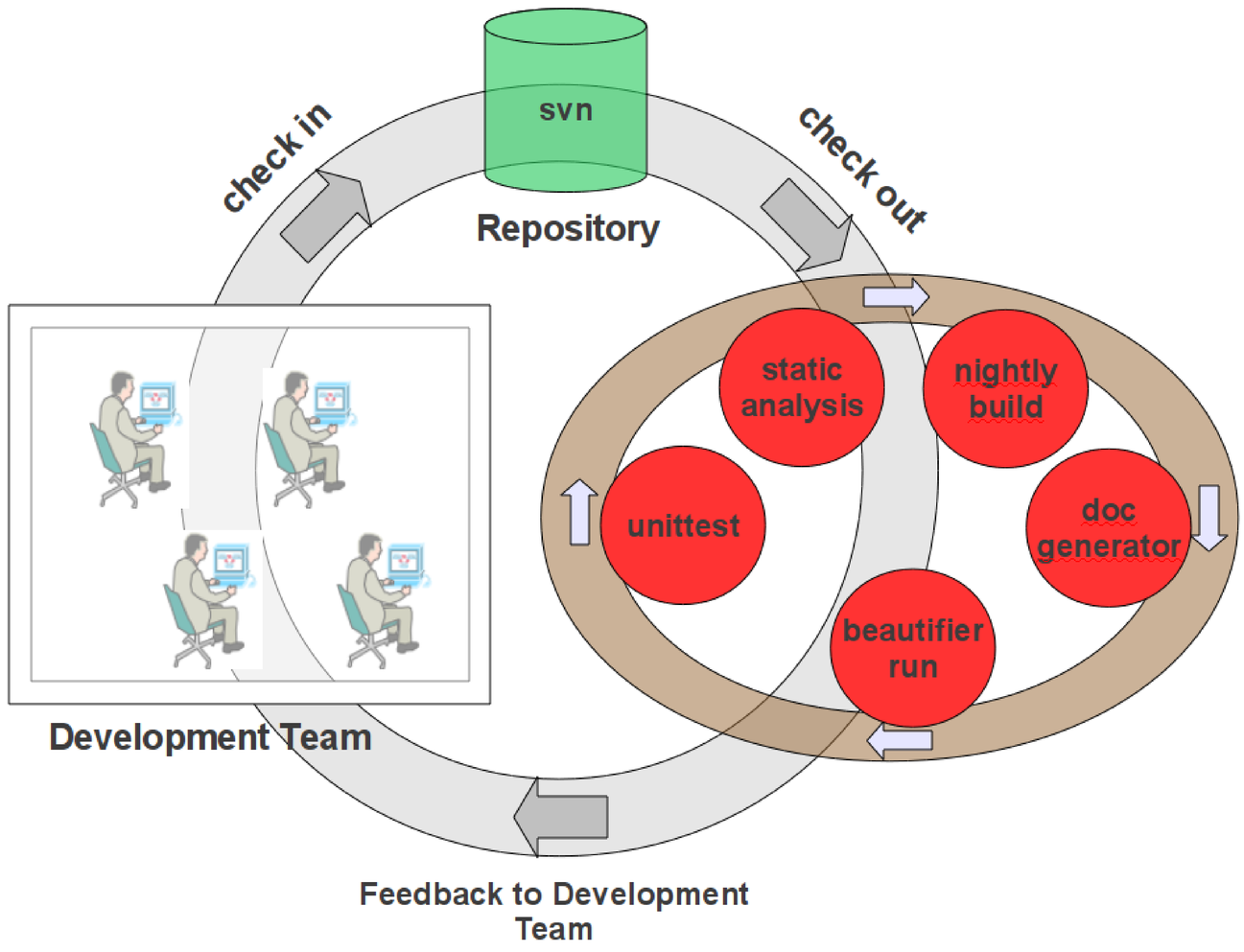
Martin Ettl[1/2], Alexander Neidhardt[1] , Walter Brisken[3], Reiner Dassing[4]
([1]FESG Wettzell,[2]MPIfR Bonn, [3]NRAO, [4]BKG)

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, 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 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 static code analysis tools. These tools produce warnings, similar to those of a compiler, but more pedantic. The reports of this analysis are translated to HTML, similar to the nightly-build reports. 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 the software to ensure quality of the projects at each development stage. These checks are now also offered as a service to the scientific community. The DiFX software correlator project already uses this offer.

What does continuous integration mean?

During the development of a software project it is hard to determine the state and stability of the current development version. Neither side-effects, nor portability issues can be detected during the development phases, especially when multiple developers work on resources, having relations to other projects. The first attack to this problem was to set up a centralized version control management system, where each developer commits his changes regularly (at least each day) to a centralized software repository. All the different versions are managed and stored in this repository, so that 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, also the newest version of the source code is always available and it can therefore be inspected intensively. Based on the newest version of the software, a collection of separated inspections run automatically, to check the source code (see red circles). Then, the results of the tests are converted to HTML pages. These results are published on the project homepage[9] in a password restricted area, to which only the developers of the project have access. Therefore, each developer can use this information to detect and fix possible problems in the code. This continuous integration work flow reduces the amount of severe issues during the whole development phases. Currently, the DiFX[2] community uses this service. The whole work flow of the continuous integration concept is shown in the figure on the right.



What is static code analysis of source code?

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[2,4,5,8] are used. These tools aimed to find bugs, which a compiler does not detect in C/C++-source code. It checks the code for memory leaks, null pointer dereferencing, unused variables, not initialized variables, mismatching allocations/de-allocations, buffer overruns, memory accesses out of bound and many more issues.

How to prepare for 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 basis can be compiled by simply calling the top-level makefile. This is done automatically every night on our Linux-servers, using several GNU-compiler versions (4.1-4.3).

Why unit tests?

Unit tests are small test programs to check the plausibility of function behavior and results on module level. For this, we have created a programming based environment to collect all these 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 operating systems to reveal portability issues. Furthermore, the test-coverage is measured using the GNU-compiler. This information about the current test-coverage as well as the unit-test report gives an overview of the current software test state. Based on this information, it is possible to measure the quality of the tested source code.

What is a documentation generator?

The developer documentation is created by doxygen[3], an open source documentation generator. This tool reads the source code, including all the comments, extracts the needed information and generates developer documentation in several formats including call-graphs and Unified Modelling Language diagrams. Running the documentation generation automatically, is helpful for developers. This especially, supports 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 beginners to understand the programming interfaces and the internal structures.

Why spell check of source code?

Due to the limited spell checking capabilities of programming editors, an automated spell checking tool[7] helps to reduce the number of misspelled words in source code and ASCII files. Finally, this improves the quality of the code an its automatic generated documentation.

Why using a source code formatter?

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

References:
[1] http://astyle.sourceforge.net/
[2] http://sourceforge.net/projects/cppcheck/
[3] http://www.stack.nl/~dimitri/doxygen/index.html
[4] http://www.splint.org/
[5] http://www.dwheeler.com/flawfinder/
[6] http://www.statsvn.org/
[7] http://git.profusion.mobi/cgit.cgi/lucas/codespell/
[8] http://code.google.com/p/nsiqcppstyle/
[9] http://econtrol-software.de/
[10] http://www.aoc.nrao.edu/~wbrisken/
[11] http://cira.ivec.org/dokuwiki/doku.php/correlator/difx