Alexander Neidhardt et al.: Ideas for a Cooperative Software Development for Future GGOS Stations, IVS 2012 General Meeting Proceedings, p.207–211
http://ivscc.gsfc.nasa.gov/publications/gm2012/neidhardt2.pdf

Ideas for a Cooperative Software Development for Future GGOS Stations

Alexander Neidhardt, Martin Ettl

Forschungseinrichtung Satellitengeodäsie (TUM), Geodetic Observatory Wettzell
Contact author: Alexander Neidhardt, e-mail: neidhardt@fs.wettzell.de

Abstract

The development of software is a creative process, which offers a huge degree of freedom. In scientific fields a lot of researchers develop their own software for specific needs. Everyone has their own preferences and backgrounds regarding the used programming languages, styles, and platforms. This complexity results in software which is not always directly usable by others in the communities. In addition, the software is often error-prone as hidden bugs are not always revealed. Therefore ideas came up to solve these problems at the Geodetic Observatory Wettzell. The results were coding layouts and policies, documentation strategies, the usage of version control, and a consistent process of continuous integration. Within this, the discussed quality factors can define quality metrics which help to quantize code quality. The resulting software is a repository of tested modules that can be used in different programs for the geodetic space techniques. This is one possible contribution to future GGOS stations.

1. Introduction

Source code is just an individual representation of an algorithm prepared for a processing platform. Such representations can have very individual forms and are expressions of the programmer. But software quality is also dependent on a clear structure, readability, and used mechanisms as errors can be hidden behind unstructured and unreadable code text, thus making it very difficult to maintain. Figure 1 shows how dense but unreadable code can be written. The program really solves the task to position N queens on an NxN chess board so that one code is not in the way of another.

```
1  int v,i,j,k,l,s,a[99];
2  main()
3  {
4    for(scanf("%d",&s);*a-s;v=a[j*=v]-a[i],k=i<s,j+=(v=j<s&&(!k&&!!printf(2+"\n\n
         %c"-(!l<<!j),"#Q"[l^v?(l^j)&1:2])&&++l||a[i]<s&&v&&v-i+j&&v+i-j))&&!(l
         %=s),v||(i==j?a[i+=k]=0:++a[i])>=s*k&&++a[--i])
5      ;
6  }
```

Figure 1. A code snippet from "The International Obfuscated C Code Contest" demonstrates how dense but unreadable codes can become [7].

Another problem is the testing behavior of developers with their source code especially in the scientific field. While the industry has dedicated certifications, scientific programs are almost untested, even if the developers perform tests. Dependencies on compilers and platforms or sketchy

function checks reduce the re-usability and even the functionality of a program. Based on this the authors of this article made a private initiative and used ideas, discussed in another article of these proceedings [5], to realize a continuous integration environment, which is also accessible for other scientists. This service is offered on a private Web site of the authors and is currently for free.

2. Continuous Integration as e-service for the Community

Continuous integration means that the newest versions of source code are downloaded from a version control system and are used for automated, daily inspections. Static code checks, nightly builds for different platforms and multiple compilers, code beautifier runs, documentation generators or even unit tests can be started automatically on the latest software version to check the included source code. The results of these inspections are converted to HTML pages. These are published on the e-service homepages [3] 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 2 gives an example of the service.

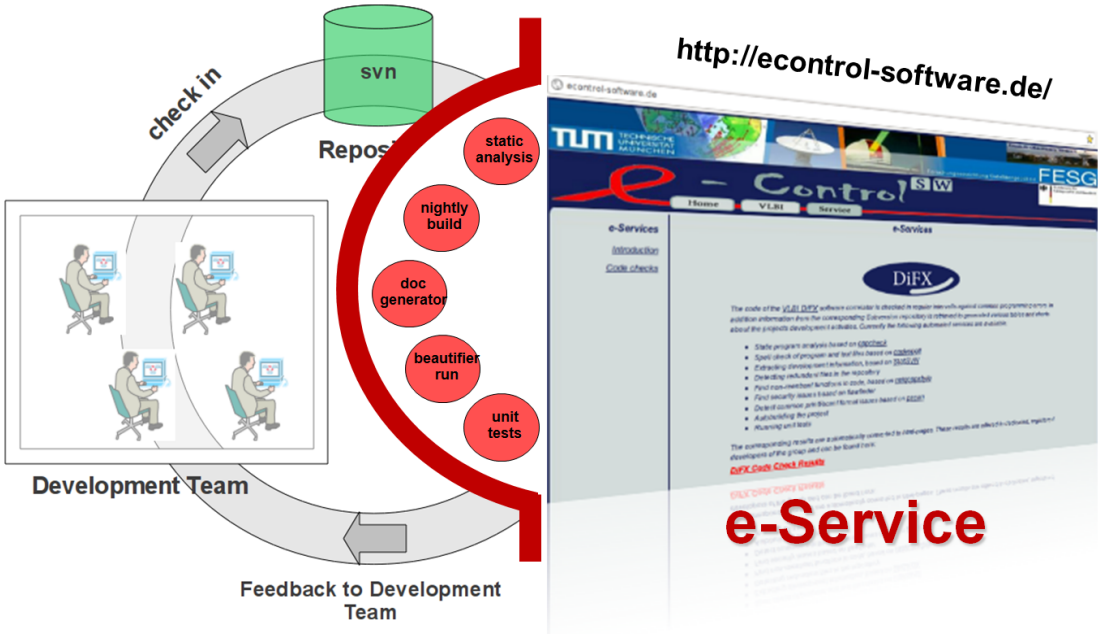

Figure 2. Continuous integration mechanisms as e-service for the science community.

Full effectiveness of the service is only given when the following items are under consideration:

- Reduction to a few software languages (to reduce needed checker tools and complexity).
- A style guide with coding layout and coding policies (to offer a unique style and become comparable between projects in terms of code metrics).
- A general way to use legacy codes (to include older codes, e.g., written in Fortran).
- Documentation generation (to get the developers documentation automatically, which can be very detailed if special identifiers and structures are used within the code).

- Automated code tests and inspections (to find errors which are not found by the compilers and to classify software quality in the sense of a quantitative metric).
- Version control (to support the continuous integration workflow with a version management).
- Offering of tested, open source toolboxes (to have reusable, well tested modules).

3. The e-service Chain of Tools

The offered environment combines several Open Source tools and concentrates on C/C++ codes (some tools can also deal with other languages, e.g., the document generator). While the tools themselves are just a loose collection, the e-service combines them into a powerful unit, offering quick-look overviews (e.g., in the form of smiley matrices, see Figure 3), well-formatted HTML outputs and statistics graphs, or to see the historical development of errors. The goal is to offer a low-level, general metric. Metrics try to classify software with quantitative parameters, which can be measured. Such parameters are not always clearly interpreted. But lines of code, found errors, found platform dependencies, number of updates, number of new lines, etc., can give a feeling about the current status of the code. The tools in the e-service environment help to acquire such parameters. The currently offered tools are:

- Static program analysis (cppcheck [2]).
- Spell check of program and text files (codespell [1]).
- Extracting of development information (StatSVN [10]).
- Find non-reentrant functions in code (nsiqcppstyle [8]).
- Find security problems (Flawfinder [6]).
- Detect common *printf/scanf* format errors (pscan [9]).
- Generate developer documentation based on Doxygen [4]).
- Detect redundant files in the repository (own development).

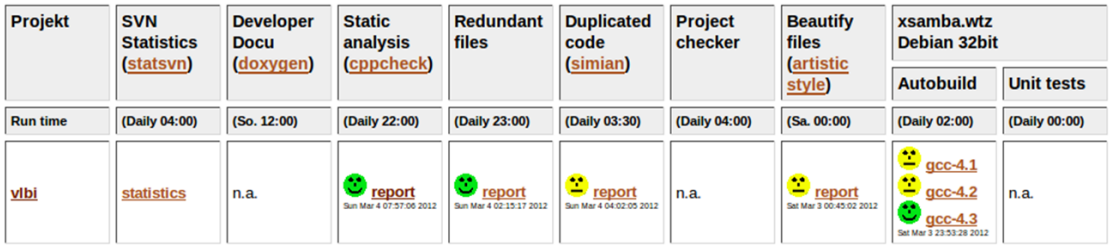

Figure 3. The quick-look overview about the project status in the form of a smiley matrix.

The inspections and checks can be run daily or weekly. The environment is also flexible enough to deal with TAR-archives if no version control repository is available. But then the workflow of continuous integration is not as effective as it might be as the response times in the development process increases.

4. Impressions from the e-service

The environment can be used to portray a certain emotion about the software status. The quick-look overview with the smiley matrix gives a first impression about the project status (see Figure 3). The statistics can be used to classify the software quality. Lines of code or found errors over time give an overview about the historic progress in a project (see Figure 4). The statistics can also be used to show the regular work in a project, for funding institutions, and for splitting the contribution to the different developers (see Figure 5).

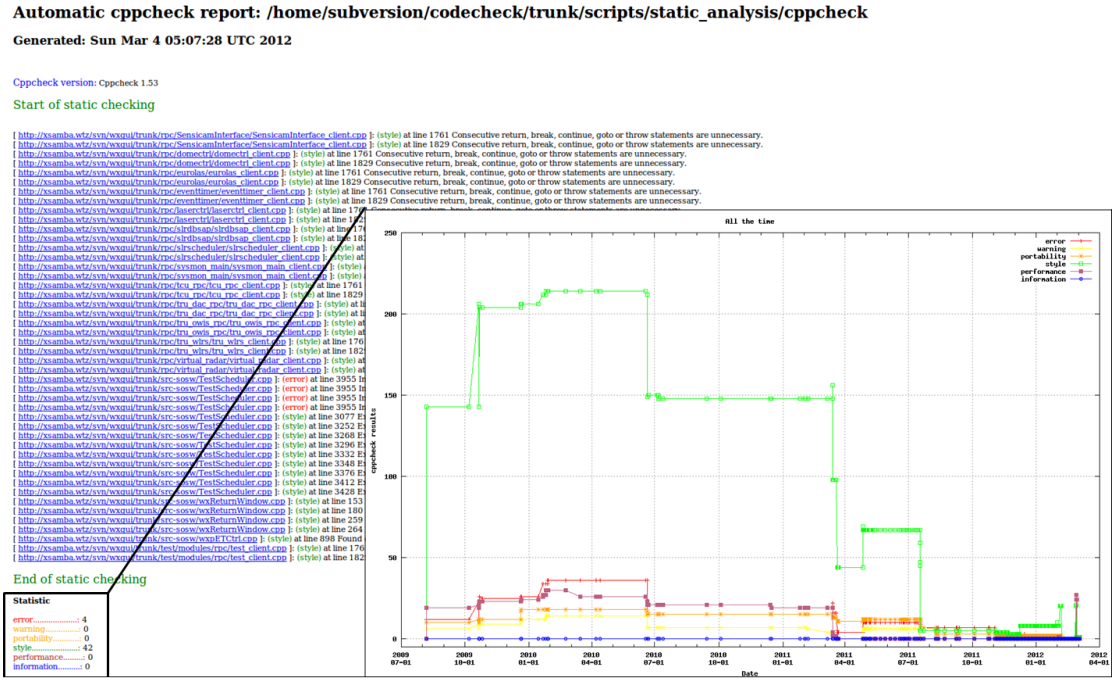

Figure 4. Statistics help to obtain an impression of the historic progress in a project over time.

5. Summary and Outlook

This continuous integration workflow reduces the amount of severe issues during the whole development phase. The e-service environment is a private initiative of the authors to help in improving the software in scientific developments and is open to others. The service is currently for free but the capacities are limited. Therefore, registrations can only be made by contacting the authors by email so that load balances and maximum transfer capacities are not overrun and to reduce costs for the authors. Currently the DiFX community uses this service on the e-Control Software Web page [3]. Also the NASA Field System is intermittently checked.

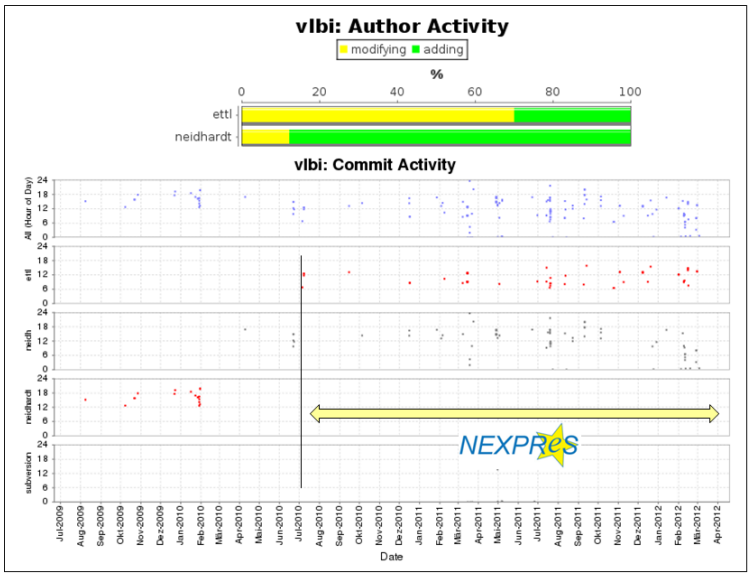

Figure 5. The contribution can be clearly allocated to developers to demonstrate the regular progress and the individual participation in funded projects.

Acknowledgements

The authors would like to thank the DiFX developer group for using the continuous integration Web environment.

References

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

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

[3] e-Control Sofwtware. http://econtrol-software.de/.

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

[5] Ettl, Martin; Neidhardt, Alexander; Brisken, Walter; Dassing, Reiner: Continuous software integration and quality control during software development. IVS 2012 General Meeting Proceedings. 2012.

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

[7] Osovlanski, Doron; Nissenbaum, Baruch: baruch.c.baruch.hint. In: Noll, Landon Curt; Cooper, Simon; Seebach, Peter; Broukhis, Leonid A.: The International Obfuscated C Code Contest. Winning entries. 1990. 7th International Obfuscated C Code Contest. 2003, http://www.de.ioccc.org/years.html#1990, Download 2011-08-04.

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

[9] PScan: A limited problem scanner for C source files. http://deployingradius.com/pscan/.

[10] StatSVN. Repository Statistics. http://statsvn.org/.