THE SOFTWARE IMPROVEMENT PROCESS – TOOLS AND RULES TO ENCOURAGE QUALITY

K. Sigerud, V. Baggiolini, CERN, Geneva, Switzerland

Abstract

The Applications section of the CERN accelerator controls group has decided to apply a systematic approach to quality assurance (QA), the "Software Improvement Process", SIP. This process focuses on three areas: the development process itself, suitable QA tools, and how to practically encourage developers to do QA. For each stage of the development process we have agreed on the recommended activities and deliverables, and identified tools to automate and support the task. For example we do more code reviews. As peer reviews are resource-intensive, we only do them for complex parts of a product. As a complement, we are using static code checking tools, like FindBugs and Checkstyle. We also encourage unit testing and have agreed on a minimum level of test coverage recommended for all products, measured using Clover. Each of these tools is well integrated with our IDE (Eclipse) and give instant feedback to the developer about the quality of their code. The major challenges of SIP have been to 1) agree on common standards and configurations, for example common code formatting and Javadoc documentation guidelines, and 2) how to encourage the developers to do QA. To address the second point, we have successfully implemented 'SIP days', i.e. one day dedicated to QA work to which the whole group of developers participates, and 'Top/Flop' lists, clearly indicating the best and worst products with regards to SIP guidelines and standards, for example test coverage. This paper presents the SIP initiative in more detail, summarizing our experience since two years and our future plans.

BACKGROUND

When LHC moved from the intense preparation and commissioning phases to operations, a consequence was increased requirements on the integrity and availability from the operations crew on the released software for controls provided for by us, the Applications (AP) section of the Controls group of the Beams department. We were at the same time facing a large and ever-growing code base, demanding more and more of our time to maintain and debug with less time to focus on new functionality. Even though some quality assurance (QA) techniques, like unit testing, were being applied in several projects, no general guidelines or standards existed. Therefore, in view of improving the quality and integrity of the products released in operations, we decided to apply a systematic approach aiming to introduce quality improvement as an integral part of the development cycle and to standardize and unify between the projects with regards to deliverables and deployment and release procedures. We call this initiative SIP, the Software Improvement Process.

OBJECTIVES

The objectives set for this initiative are:

To think of and organize us as one big team, not many small ones. This means that everybody and nobody own the software produced by the section. It should therefore adhere to the standards and guidelines agreed by all of us, not follow the personal preferences of one developer. This is important in an environment where many developers collaborate on the same software and where the turnover resulting from short-term contracts are fairly high.

To achieve better quality of products that is measurable based on predefined metrics and with an agreed set of deliverables. Metrics are important as they give us the means to measure progress, which helps encouraging the developers to apply the standards and guidelines.

To reduce code base growth by promoting the development and use of common frameworks, libraries and components, avoiding duplication.

To provide better and more comprehensive documentation of the process and components.

To achieve a better software production process through incremental improvements. We don't claim to have all the answers as we start therefore we will adapt as we go and as we learn what works and what does not work, following the evolution of the industry recommendations and tools available to us.

The process focuses on three areas: The development process itself, the QA tools available to automate the process as much as possible, and how to encourage developers to include QA in their everyday work.

DEVELOPMENT PROCESS AND TOOLS

In the AP section we apply a systematic approach, a development process, to ensure a timely delivery of software corresponding to the clients needs and requests while ensuring improved productivity and software quality. It is an iterative process, where for each iteration the project goes through the stages 'Design', 'Implement, Test and Document', and 'Deploy and Maintain' as depicted in Figure 1.

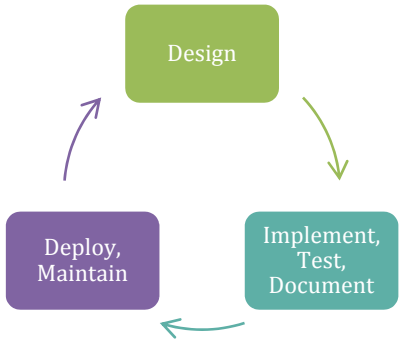

Figure 1: The stages of the development process.

For each of the stages depicted in Figure 1 we have in SIP defined the recommended or mandatory activities and deliverables. We have also identified tools that will help us automate the process as much as possible and agreed on their configuration. To ensure that each developer has these tools available and uses the same configuration, they have been integrated into an AP-specific distribution of our recommended IDE, Eclipse [1].

Design

In the design stage we have agreed to do more design reviews for new and existing projects. The purpose is twofold: firstly, it verifies the soundness of the design and propose improvements at an early stage of the iteration, i.e. before the developers invest much time in the actual coding and testing; secondly, it promotes knowledge sharing and collaboration between different development teams in the AP section, and help identify overlapping functionality, in view of reducing redundancy from our existing code base.

These reviews are at the detail level of sub-components. To discuss the design, we use UML class diagrams with the main classes and design patterns, and sequence diagrams with the interactions between these classes.

Implement, Test, and Document

In this stage the SIP focuses on three areas: the code, the documentation and the testing.

Code reviews go into more detail than design reviews. They aim at finding bugs, at making sure the code is maintainable and at verifying that our development conventions are met. However, as code reviews are very time consuming, we have decided to focus only on the most critical parts of our code (e.g. core libraries or multi-threaded code) and on code written by junior developers that need mentoring. This is done in an interactive way with person-to-person reviews of the code, but also in a lightweight, offline manner relying on the Atlassian tools FishEye+Crucible [2] integrated with Eclipse. Through this tool a committer can set up a review for a change-set, invite a number of fellow developers, which are then notified via email and can review the code changes in Eclipse and comment inline.

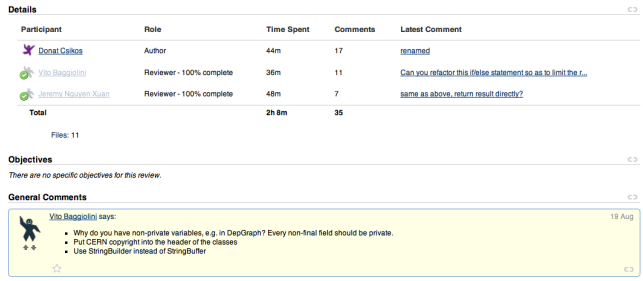

Figure 2: Code review using FishEye+Crucible.

In addition to code reviews by humans, we rely on static code analysis tools to automatically spot the most common mistakes and bug patterns. In the beginning of the SIP we performed an investigation and comparison of several tools and agreed on using FindBugs [3] and Checkstyle [4]. Plugins for these tools are distributed with our tailored Eclipse distribution, and we have agreed on a common configuration for each, also distributed with the Eclipse distribution. In addition to these external tools, we have settled on a common configuration of the Eclipse warnings, also pre-configured in our Eclipse distribution.

The benefit of having the tools integrated with the IDE is that they show up as other compilation errors, giving immediate feedback to the developer about potential problems.

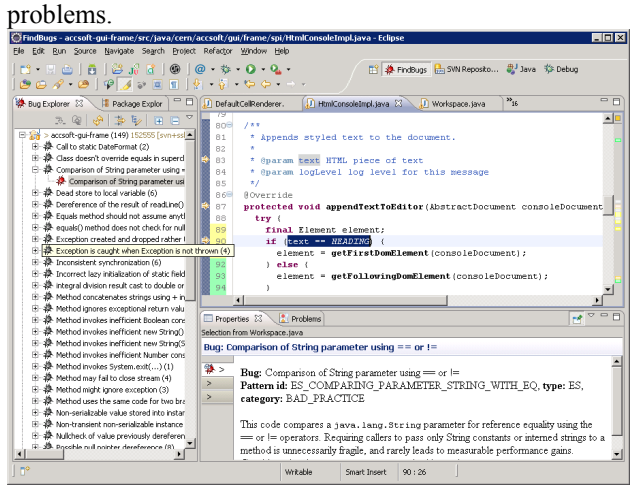

Figure 3: FindBugs report in Eclipse.

Another area the SIP focuses on is the level of testing in the projects. Even though most developers agree on the benefits of unit testing, not all take the time to implement unit tests as new functionality is added. To improve on this situation we have agreed to make unit tests a mandatory deliverable of a project: a minimum coverage of 30% for non-trivial classes must be achieved before a project can be released. We use Clover [5] from Atlassian to check the level of unit test coverage and again there is a plugin integrated with our distribution of Eclipse, giving immediate feedback to the developers of the level of coverage and the high-risk classes. Tested code appears in green, while untested code is highlighted in red (c.f. Figure 4).

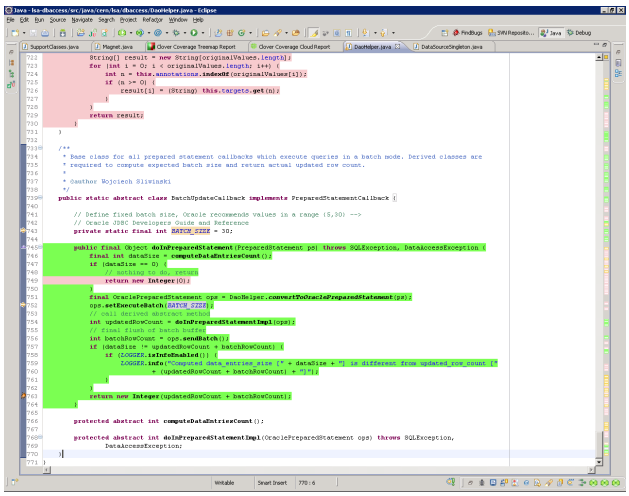

Figure 4: Test coverage indicated using Clover.

To make sure that changes in one project do not break other projects that depend on it, we have put in place a Continuous Integration (CI) server using the Atlassian tool Bamboo [6]. Whenever a developer commits changes to the source code repository, this tool checks out the new sources, compiles them and runs the unit tests. It then does the same with all dependent projects, in a cascading way, to assure that everything still compiles and all the unit tests still succeed.

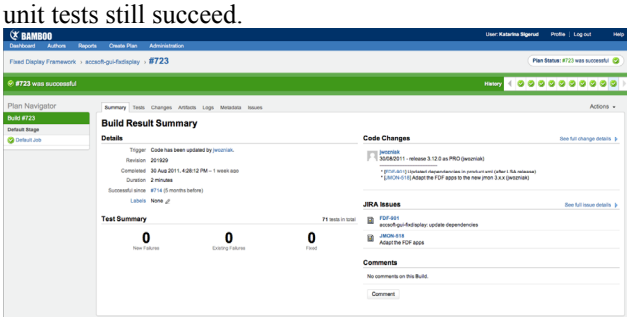

Figure 5: Bamboo build plan summary.

Documentation in SIP concerns two areas: to document the process itself and as a project artifact mandatory before a release.

To document the process, we have put in place a wiki page, detailing the set of project artifacts, best practices and standards that we have agreed on. It also lists the tools we have decided to use and their configuration.

Regarding documentation as a mandatory artifact of a project, we believe that for the documentation to be kept up to date, it should be as close to the code as possible. We therefore rely in first place on documenting the code, using the Javadoc [7] tool. At least for the base java source package(s), there should be a clear description in a file called *package-info.java,* summarizing the functionality of the package and sub-packages. As needed and for more details regarding specific sub-packages, there can be one *package-info.java* per sub-package.

Regarding documentation inside the code, we have agreed on common file and class headers, containing items like the copyright statement and SVN variables.

At least all public classes and interfaces must be documented with Javadoc following the agreed guidelines, and there should be Javadoc links to other packages (e.g. JDK).

We have also agreed on a common code formatting, available by default in our Eclipse distribution.

Both the Javadoc and the code formatting are checked using Checkstyle.

For all documentation that cannot be done using Javadoc, we are relying on Atlassian's wiki Confluence [8] to document the process and project-specific information.

Deploy and Maintain

In the deployment and maintenance phase, we have focussed on introducing a common build, release and deploy procedure using tools developed in-house. We have made a particular effort to standardize the deployment of Java server-side processes. For this we have agreed on a common naming, location and directory structure, supported by tools that enable us to easily deploy a new version of our products into operations, but also to roll-back to the previous version if necessary. The benefit is that the processes are now easily recognized and located by most members of the AP section, and allows them to intervene on processes of their colleagues, e.g. to restart or roll-back a process if necessary.

Once a process has been deployed operationally, the follow up of issues and new requests is important. We have chosen to rely on Atlassian's issue tracking tool JIRA [9] for this as it gives us the transparency we are looking for and is easily configurable to our needs. Being an Atlassian tool it also integrates well with the other tools in our development process, like the CI server Bamboo and FishEye+Crucible for code reviews where an issue number is the traceable item across all three, as shown in Figure 6.

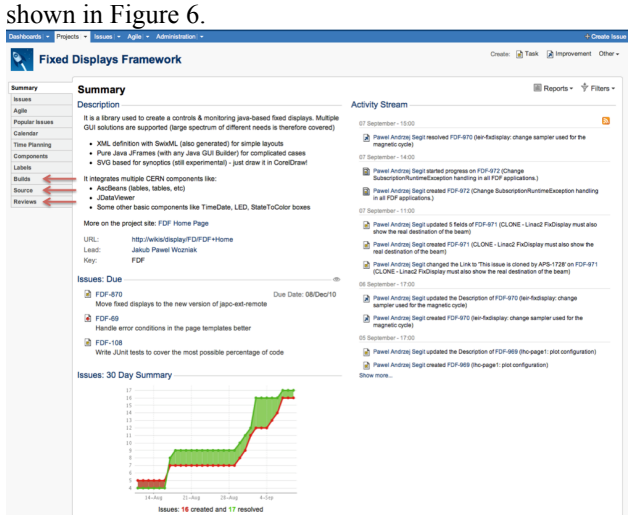

Figure 6: JIRA project summary page with links to Bamboo builds and to sources and reviews through FishEye+Crucible.

CHALLENGES

The two major challenges of SIP have been (1) to agree on the common standards and configurations, such as the common code formatting and Javadoc documentation guidelines described previously and (2) to encourage the developers to do QA.

For the first point, the approach we have taken is summarized in the first objective described above: *Think of and organize us as one big team, not many small ones*. As the code belongs to the section, not to a project or an individual developer, it should adhere to the guidelines and standards agreed by everybody not the preferences and habits of a single person.

For the second point, most members of the AP section saw the benefits of the QA techniques discussed here, and many projects already applied some of them consistently. However, they felt that their priority was to sort out issues and to provide new functionality requested by the users, and hence, they felt they did not have time for more QA work. To address this issue, we have decided to 'officialise' the time spent on QA and make the artifacts it produces mandatory deliverables of a project before release. QA objectives are part of a project's yearly objectives and prioritized and reported on regularly as with more traditional objectives. The progress for each project to adhere to the agreed guidelines is also tracked and 'Top/Flop' lists are presented on our wiki showing the best and worst projects. Finally, we have introduced "SIP days", approximately one every 3 months when everybody in the section works on a common goal, e.g. to increase the test coverage, to complete documentation or to standardize deployment configurations. This is to encourage the developers who might still have difficulties to find the time for QA activities to apply the guidelines, standards and tools in their work.

FUTURE PLANS

As for our development, we are also for the SIP applying an iterative process and continuously considering how to improve and move forward with QA related work.

Inside the section, there is still some work to finalize the list of guidelines but the main focus will be on improving the tool integration and the tracking of the progress through Top/Flop lists. As well as developing our own tool, we are also considering tools like Sonar [10], a platform to manage code quality that integrates many of the tools mentioned above, displaying the results in a clear manner.

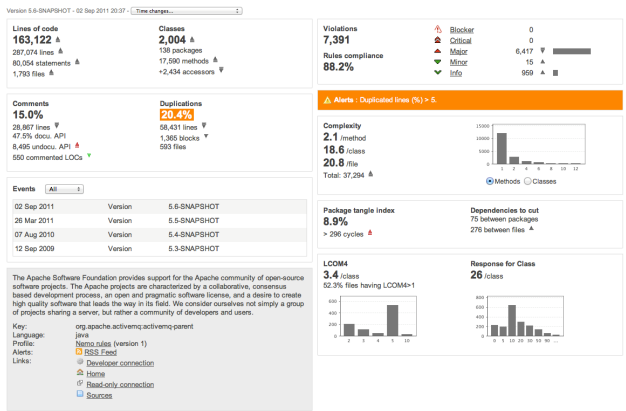

Figure 7: The Sonar dashboard.

At the group level, there is also an increased interest in QA, most particular in extending the SIP principles to C/C++ projects.

REFERENCES

[1] Eclipse: http://www.eclipse.org/.
[2] FishEye+Crucible: http://www.atlassian.com/software/fisheye, http://www.atlassian.com/software/crucible
[3] FindBugs: http://findbugs.sourceforge.net/
[4] Checkstyle: http://checkstyle.sourceforge.net/
[5] Clover: http://www.atlassian.com/software/clover/
[6] Bamboo: http://www.atlassian.com/software/bamboo
[7] Javadoc: http://www.oracle.com/technetwork/java/javase/documentation/index-jsp-135444.html
[8] Confluence: http://www.atlassian.com/software/confluence/
[9] JIRA: http://www.atlassian.com/software/jira/
[10] Sonar: http://www.sonarsource.org/