ASSESSING SOFTWARE QUALITY AT EACH STEP OF ITS LIFECYCLE TO ENHANCE RELIABILITY OF CONTROL SYSTEMS

V. Hardion, A. Buteau, N. Leclercq, G. Abeillé, S. Pierre-Joseph Zéphir, S. Lê
Synchrotron Soleil, Gif/Yvette, France

Abstract

A distributed software control system aims to enhance the upgradeability and reliability by sharing responsibility between several components. The disadvantage is that it makes it harder to detect problems on a significant number of modules. With Kaizen in mind we have chosen to continuously invest in automation to obtain a complete overview of software quality despite the growth of legacy code.

The development process has already been mastered by staging each lifecycle step thanks to a continuous integration server based on JENKINS and MAVEN. We enhanced this process, focusing on 3 objectives: Automatic Test, Static Code Analysis and Post-Mortem Supervision.

Now, the build process automatically includes a test section to detect regressions, incorrect behaviour and integration incompatibility. The in-house TANGOUNIT project satisfies the difficulties of testing distributed components such as Tango Devices.

In the next step, the programming code has to pass a complete code quality check-up. The SONAR quality server has been integrated in the process, to collect each static code analysis and display the hot topics on summary web pages.

Finally, the integration of Google BREAKPAD in every TANGO Devices gives us essential statistics from crash reports and enables us to replay the crash scenarios at any time.

We have already gained greater visibility on current developments. Some concrete results will be presented including reliability enhancement, better management of subcontracted software development, quicker adoption of coding standards by new developers and understanding of impacts when moving to a new technology.

INTRODUCTION

Building, operating and maintaining a control system are complex operations. When the SOLEIL Control Group chose Tango as the distributed control system for the SOLEIL synchrotron, they firstly thought about upscaling good practices from previous laboratories to a big facility. De facto, a distributed system allows:

- Sharing between programs, as opposed to monolithic applications.
- Load-balancing to scale the number of devices to control.
- A standard communication protocol to focus development on the "business" code.

A new technology can afford the adoption of good practices only if we consider that resistance to change is an important task to manage. While we didn't call the acceptation of Tango at Soleil Kaizen, it was.

Our Kaizen : A Lean Quality

The main objectives at SOLEIL are to improve productivity and quality. Our Kaizen is inspired from the "Toyota way" or "Lean", meaning that rapid production is not possible without managing quality, and vice versa. The benefit is to detect any problems as soon as possible in the software lifecycle to reduce the cost of resolving them, which increases exponentially as production progresses.

Moreover our client has asked the Control Group to supply just-in-time solutions to their problems. So we have managed this quality project like an agile project focused on developer productivity with limited time resources.

This philosophy is based on good practices revealed by the open source community and our own experience:

- "Bottom-Up": Only the developers know how to work better. The quality process aims to generalise isolated good practices
- "Agility": We prefer modify our tools to prevent us from the non-quality instead of write documentation.
- "Don't Repeat Yourself" or DRY: We want to eliminate all small repetitive actions with no business value for the developer's job.
- "Keep It Simple, Stupid" or KISS: We won't specify our quality like a "silver bullet" with "tunnel effect" but rather by building slowly by small iterations (lasting weeks) from the actual requirement and supplying new automatism, new monitoring, new checks ASAP.
- Standardisation: Priority on components that follow our standard which can benefit from all advantages of the system. All deviations are clearly identified in our Wiki.

Only when the tools can't make a rule transparent for the developer, writing reference documentation is mandatory. But experiences show that maintenance time increases as there are many interpretations of each quality document.

Software Factory With Continuous Integration

So our quality system is mainly embodied in a Software Factory deployed in early 2008. This system integrates all tools and automation to build and monitor each piece of software. The functional perimeter includes registering new projects, building, testing and integration for deployment. The installation of the Continuous Integration principle [1] has rationalised much of the Control group's business process. This setup allowed time to be leveraged to focus on quality.

With a "just-in-time" job scheduler like Jenkins [2], the aim was to deal with automated actions at the most appropriate time. This is the case with a change in the

Copyright © 2011 by the respective authors — cc Creative Commons Attribution 3.0 (CC BY 3.0)

source code that triggers a compilation job. It's more convenient to debug code from one change than a nightly build which will compile several changes.

All our software components follow the same lifecycle defined in Figure 1, where each step is associated with a quality check.

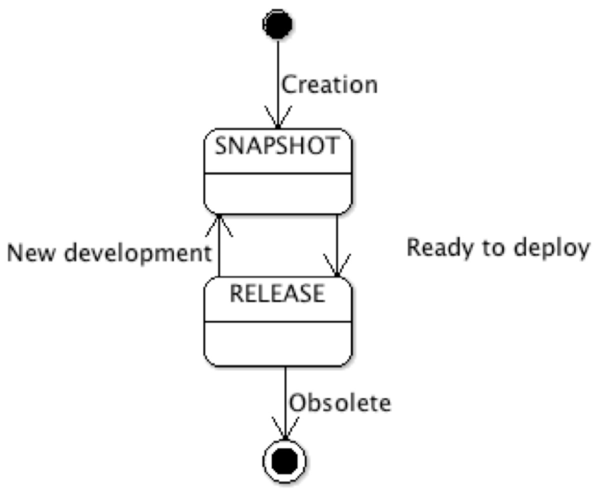

Figure 1 : Lifecycle.

In this kind of system, every new process or new tool should have the ability to be non interactive. This system is quite stable because the developers prefer to benefit from its advantages rather than use non standard tools. Even if they choose a new tool, the quality project could integrate new requirements.

The sections below will describe the extension of the Software Factory. With this new version, we chose to focus on 3 main axes: Automatic test, Continuous Quality Control and Monitoring production life.

AUTOMATIC TEST

The tests can guarantee the expected behaviour for end users as this is an important part of final quality. Although this could be the most important axis, it's also the most difficult to apply:

- No transparency: Developers have to write automatic tests themselves.
- Technically: Some difficulties with Client/Server paradigm or Graphical software
- Hardware: Equipment is often not available for automatic tests.

TangoUnit : Test For Tango Devices

TangoUnit aims to reproduce an environment for software that integrates Tango Devices. With TangoUnit, the goal is to supply a small framework to abstract registration, execution, deletion of Tango devices when the developer creates their testing environment. Tests using TangoUnit are considered as part of Integration Tests.

Simulation

An analysis made in 2010 on some beamlines reveals that only ~20% of deployed devices are directly connected to equipment. Others are process devices or from a higher layer based on the hardware equipment. It means that simulation should only consider equipment devices. This enables integration tests to be implemented for processes (high layer) devices. In the same time, some simulated devices have been created and one generic device called Transformer allows the behaviour to be changed dynamically to completely mock a real device.

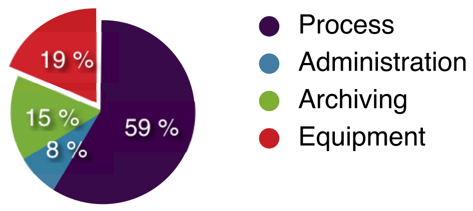

Figure 2 : Device usage.

For the ~20% of low level devices, an internal simulation mode is necessary.

Experience

How do the unit tests help to manage the recent reorganisation of projects responsibilities?

When S.Pierre-Joseph Zéphir, an ICA software engineer, took over the responsibility of the supervision project [11], she knew a little about the internal organisation of source code and the functionalities already implemented. On the other hand, the users needed to retain confidence in the one of their main tools to monitor the machine and the beamlines. If you added the current stabilisation of the new underlying graphical library and the migration of legacy components, you obtain an "explosive cocktail".

The right way to guarantee stability was to invest in unit tests especially for graphical software. Thanks to Ordinal [3], the editor of the supervision framework, which brings expertise with JFCUnit [4] a graphical testing tool, the feasibility was quickly assessed. By successive iteration, the acquired experience allowed us to better understand these black box tests on components used directly from final users.

This project cost 80% of the time spent for enhancing the tool over a period of 4 months. Initially difficult to estimate, the Return On Investment gave us more confidence in the user trust level and the comfort of portability.

A good side effect is that the rest of the team also benefited from integration tests, because the supervision software is one of highest in the dependency tree.

Next Steps

The adoption is very slow but for new project. We learned a lot from our initial success about how we can

Copyright © 2011 by the respective authors — cc Creative Commons Attribution 3.0 (CC BY 3.0)

benefit from major changes to implement tests in a project.

Today some Java projects use JUnit and a few JFCUnit for graphical tests. Some Tango Devices have a little test coverage with TangoUnit. But automatic tests are not the default practice and we will have to interest developers by training or with friendly tools.

STATIC CODE ANALYSIS

Continuous Quality Control is a process which aims to evaluate the compliance of projects to the Control Group's Quality Assurance criteria. One category specifically addresses the issue of the code quality, this is Static Code Analysis. Java developers are required to define a standard way to write code, from style to good practices.

The difficulty of obtaining a complete overview of all modules, understanding the metrics and to determining priorities from the huge metrics almost caused this project to fail.

Sonar

Sonar is an Open Source Software (OSS) developed and supported by SonarSource [5]. It aims to analyse the quality of components and report on them with a web server. The main functionality is to aggregate metrics to show only the essential data, with the possibility of monitoring metrics trends. It comes with preconfigured compliance levels for each rule.

Each new release of component triggers a complete analysis (see Lifecycle). The developer can also trigger an analysis during the development phase to allow them to anticipate the quality and correct it before the Release.

Java

Today, the Java side has been in production since late 2010 and some critical projects are actively monitored with:

- the dependency structure provides information about the abstraction level,
- the code duplication information gives maintainability level,
- the highest level violations help focusing on possible bugs.

C++

Sonar has no built-in functionality for C++. Thanks to the plugins system, it had been possible to extend Sonar. In this context, we studied the OSS market place with these constraints: multiplatform, easily parse report, no false positives and standard in the community [6]. Some tools stand out without reaching the level of Java tools.

Today the C++ analysis is integrated to Sonar through the CXX extensions co-developed by Soleil. All results come from CppCheck [7] for bug detection and Vera++ [8] for the syntax and the style of coding analysis. But we still have to invest time in determining the compliance level of each rule.

Experience

When Synchrotron Soleil decided to update the Java implementation of Tango device, G.Abeillé, the ICA engineer who has been in charge of this project, was able to check that her implementation was compliant with the OSS standard and that her unit tests were efficient.

Tango defines a standard protocol for communication between Servers and Clients. A Tango Device has some complexity with the number of execution paths with the different input and output types.

Although unit testing has a cost, we can monitor the code coverage thanks to Sonar, associated with the measure of the Cyclomatic Complexity [9]. Thus to know the effective coverage allows us to reduce the number of unit tests to the most efficient level. Others metrics like number of comments, duplication of code lines, rules compliance was useful to be OSS compliant.

Unit tests cost 2 weeks compared to the 2 months of code phase. Subsequently the unit testing cost could have been integrated in the initial time if they were written first.

Next Steps

Now the experience has benefited other projects to enhance the maintainability or recently to choose subcontractors who can comply with new ICA best practices. Sonar is also ideal to help the integration of young software developers with an accurate explanation of all rules and associated good practices. We are trying to define the monitoring process for all projects with global metrics but in an efficient way with the "Sonar Views" plugin.

POST-MORTEM MONITORING

Crash Reporting

Certainly the third success experience of this article. N.Leclercq, who is in charge of the Machine control system, enhances the quality by heading the Crash Reporting project, an "accelerators post-mortem"-like system but for Tango Control System. This process, which has been set up in production since late 2010, was motivated by the difficulty in debugging low occurrence and non-repeatable software crashes. Our accelerators operators didn't use to report these kind of events so our statistics were poor on it.

The only valuable solution was to invest in a Crash Reporting system and after evaluating the market we chose Google BreakPad [10] as the only open source and multi-OS implementation. By monitoring the current threads with another static thread able to catch all exit events, its operating principle is really non-intrusive. This library is encapsulated into the in-house CrashReport library to adjust our own parameters, such as output format or to retrieve some Tango Device informations. Then main 3rd party libraries are compiled with this CrashReport library. With the software factory, we added small pieces of code in main.cpp file of all Tango Devices used at SOLEIL. After deploying a new version of

Copyright © 2011 by the respective authors — cc Creative Commons Attribution 3.0 (CC BY 3.0)

common build parameters, the activation was carried out for all libraries and devices.

Crash Assessment is computed with each log of each Tango Device. The developer is able to replay the context thanks to the associated debugging information, although the programs are compiled with optimised options, mandatory for production deployment. Also no code instrumentation is necessary.

Quick Win

Not straightforward. This project cost approximately 80 man hours mainly due to the lack of documentation of the BreakPad project. But the ROI was immediate with working log and objective feedback each time the origin and type of failure was questioned. With this process we solved the obvious crashes caused by 30 out of the 300 Tango Devices.

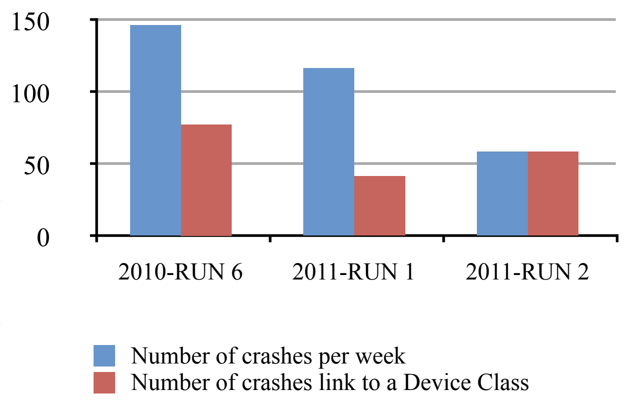

Figure 3 : Software crashes figures per Beam run (Accelerators and Beamlines).

Now we have stabilised the deviation and a maximum of crashes. Graph reports bottom out the natural law where as 80% of the crashes were caused by around 20% of the devices which corresponded with the maintenance of the global Tango Device legacy. The Crash Reporting has increased the quality of production such that the Machine's staff has clearly seen a "skyrocketing" progression.

Next Steps

This process is well monitored and maintained, but some manual operation are costly. Actually the report is made globally by hand. Other points also consist of the replay of crash context that needs the debugging info supplied with the compilation. The size of the whole distribution of binary was 5 times bigger than before. In fact, the evolution of this process will eventually be centralised like Firefox is with the Socorro project in which all crash notification will go to a server that also keeps the debugging information.

CONCLUSION

Besides of these main axis, several others monitors developed at Soleil gives us quick win results :

- Report project in activities (i.e in snapshot Status) notify developers about forgotten to release theirs projects.
- Report failed projects from dependency change is useful to analyse impact for any library evolution. This report had helped a lot to identify blocking point when we migrated all our component to the new version of Tango 7.
- Report on versions changes from the last official deployment. This report is transmitted to users to show where are the risks for the next deployment.

Obsolescence

Another point in progress we have been working on is how to target the unused Java components from the supervision software written by end-user [11]. These old components carry a lot of weight in major evolution of software. Here a simple graph analysis had allowed us to focus unit test only on used component and didn't waste our time. Cleaning the legacy should be valuable.

Human Touch

The position of the quality software manager implies good knowledge as well in software development to understand the requirement of developers than in system administration to deal with software installation. This allows to be agile with requirements.

It's important to imply the developers to these software quality processes. Sharing knowledge is also necessary for the communication and the dynamic of the team. For this purpose, we organise each month an internal regular meeting for software developers . These so-called "Café Java and C++" aims to create a dynamic for the continuous improvement of our software developments.

REFERENCES

[1] Making Continuous Integration a Reality for Control Systems on a Large Scale Basis, A. Buteau, S. Dupuy, V. Hardion, S. Le, M. Ounsy, G. Viguier, SOLEIL, Gif-sur-Yvette, ICALEPCS'09
[2] Jenkins : http://jenkins-ci.org/
[3] Ordinal : http://www.ordinal.fr/
[4] JFCUnit : http://jfcunit.sourceforge.net/
[5] Sonar : http://www.sonarsource.org/
[6] C++ Tools : http://docs.codehaus.org/display/SONAR/Cover+new+languages+with+Sonar
[7] CppCheck : http://cppcheck.sourceforge.net/
[8] Vera++ : http://www.inspirel.com/vera/
[9] Cyclomatic Complexity : http://en.wikipedia.org/wiki/Cyclomatic_complexity
[10] BreakPad : http://code.google.com/p/google-breakpad/
[11] How to Use a SCADA for High-Level Application Development on a Large-Scale Basis in a Scientific Environment, Katy Saintin, Vincent Hardion, Majid Ounsy, SOLEIL, Gif-sur-Yvette, ICALEPCS'07

Copyright © 2011 by the respective authors — cc Creative Commons Attribution 3.0 (CC BY 3.0)