THE CSS STORY

M. Clausen, J. Hatje, J. Penning, DESY, Hamburg, Germany

Abstract

Control System Studio (CSS) is designed to serve as an integration platform for engineering and operation of today's process controls as well as machine controls systems. Therefore CSS is not yet another replacement of existing operator interfaces (OPI) but a complete environment for the control room covering alarm management, archived data displays diagnostic tools and last not least operator interfaces. In addition we decided to use CSS as the platform for the whole engineering chain configuring EPICS based process control databases, configuring and managing the I/O, editing state notation programs, configuring role based access rights and many more. Due to the ease of use of CSS as an Eclipse based product, we decided to use the CSS core also for all our stand alone processes. This helped us to reduce the diversity of running products/ processes and simplified the management. In this presentation we will describe our experience with CSS over the last two years. How we managed the transition from old displays to new ones, how we changed our alarm/ message philosophy and last not least which lessons we learned.

INTRODUCTION

The development of a new tool set was initiated by the new project at DESY – the European XFEL. This project will lead us through the next decades to come. All existing hard and software for cryogenic and utility controls had to go through a verification process to decide whether components can pass 'as is' or whether they have to go through a process of refurbishment or even new design.

For the existing operator tools it was soon clear that they have to be replaced by a new tool set.

NEW TECHNOLOGIES FOR NEW OPERATOR TOOLS

To prepare the new tool set for the future it was decided to use Java as the programming language. The next question to be answered was: Which development environment tool shall be used? This covers also the question about the core technologies to be used. Two candidates were investigated: Eclipse and NetBeans.

At the time when Eclipse 3.1 was launched it was chosen for the CSS core technologies. Many more basic technology decisions had to be made. These were identified in a workshop about three years before the first beta release of CSS.

INITIATING COLLABORATION

As a result of the workshop the CSS collaboration was started between the initial partners DESY and SNS. In addition two companies were involved in the exploration of new technologies. But what is the best way to explore the new fields? How can one set up a contract with a company when the details cannot be specified? We had to walk on new ground with respect to contracting and project management. New techniques known from extreme programming were used to setup and run these development projects.

WATERFALL OR XP?

This is not a question any more. Waterfall specifications are by definition always out of date and are outdated. But which kind of extreme programming shall be followed? A very basic concept is working in iterations. Iteration steps are small they last only a few days or weeks. The work to be done in these steps is defined in tasks. The results are checked after each iteration step. Especially when new technologies must be explored it is important to stay in close contact and verify that the team is still on the right path. Waterfall – or long iteration steps will fail in this respect.

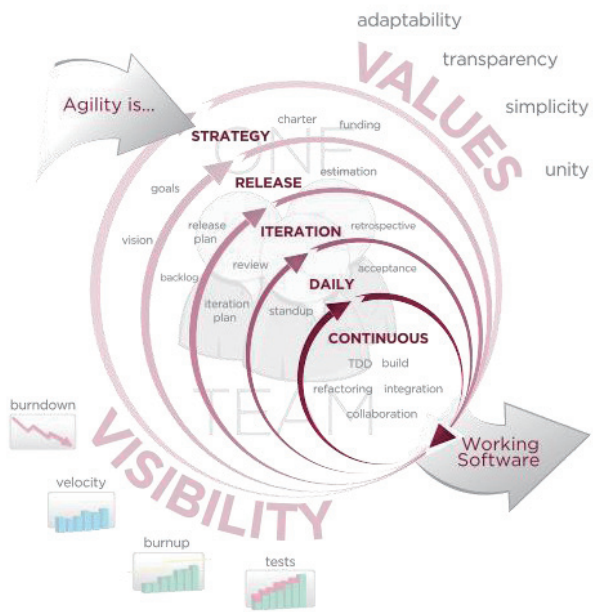

Figure 1: The Agile Methodology ©

ISBN 978-3-95450-124-3

Identifying Risk Factors

Risk factors first – this is one of the concepts for the iteration steps. This will make sure that those tasks which are critical for the whole project will be dealt with at first. Other tasks will run on a lower priority and will be solved later. In our case we had to check whether the GEF (graphical editing framework) will be adequate for the new operator interface. Several criteria needed exploration: Will it be possible to use GEF not only in edit mode but also in runtime mode? What will the performance of GEF dynamic widgets be? (several thousand updates per second were specified) And – can the layout created in the GEF editor be stored in XML and read back in?

All of these questions were checked and answered in several iterations throughout the development process. Each of these was not fully implementing the full functionality of the final product but just enough to verify the proof of concept of the individual step. All of them worked as a cut through the technology layers proofing that GEF can be used as the backbone for the new graphical user interface for the operators.

CHANGE MANAGEMENT: PATCHWORK OR POPPER?

During the course of software maintenance and change management one will experience several different kinds of approaches how changes get implemented.

Patchwork

If software changes are integrated 'on the fly' into existing code they will most likely be not well integrated. Many changes will in the end destroy the initial structure of the code base. A patchwork of adjacent code snippets will also make testing difficult – if not impossible.

It will be very likely that such code will not be accompanied by the required JUnit test cases. In some cases such code might even break existing functionalities.

Popper

A well structure code will ease modifications of the code. Even functionality/ code extensions will be easy to integrate when well defined interfaces (popper-alike) are foreseen in the design. Well defined interfaces are also mandatory for good test coverage with JUnit test.

TEST TEST TEST

Test Driven Development

Writing the test code before the production code gets written is the ideal world but the real world differs from that. If this order does not work it is important to keep in mind – and actually write – the test code afterwards. Writing JUnit test code is initially as popular as writing documentation. As soon as the benefit becomes obvious it will be written as an integrated part of the application code itself.

Healthy Collaborative Pressure

Any chain is as strong as the weakest link. This phrase is also true for software packages. The bigger these packages get the more complicated it gets to diagnose errors and potential problems. Developing code in collaboration with others will put a healthy pressure on all developers to provide stable code which does not interfere with anybody else's implementations.

Dependencies with 3rd Party Code

If your code is dependent from other code which is not developed in-house it is mandatory to 'protect' your own code base from that code by adding JUnit tests on your side to detect problems when new versions of that code (which you do not support yourself) gets committed.

CONTINUOUS INTEGRATION

Especially in a collaborative/ distributed development environment it is necessary to check whether your final product can still be built taking all of the latest commits into account. A special server is set up at DESY and on some of the other collaborator's sites to perform this job. It is continuously checking in all changes from the code repository. The new code and all of the existing code will be compiled. This method will identify code that does not compile and also incompatibilities between different packages. Ideally your own changes will be committed after you have checked your code with the latest core code from the collaboration.

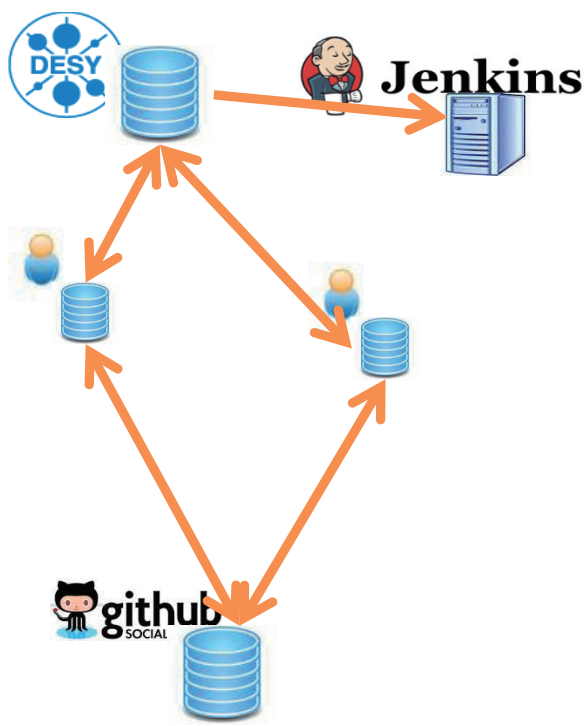

Figure 2: Continuous Integration on a Jenkins Server @ DESY.

ISBN 978-3-95450-124-3

CONTINUOUS TESTING

The second step after the continuous integration of software changes is to check for runtime errors. JUnit test are meant to check for errors beyond compilation. While most of the JUnit tests are typically started manually in everybody's own environment the intent of 'Continuous Testing' is to run these test on the same server which performs the tests after each continuous integration test. Setting up these tests on a server platform is a complex task especially when database servers or control system protocols are involved. But – there is no doubt – any time invested in testing and integration pays off in double in time which is NOT spent in later debugging and maintenance and in the end unhappy end users.

THE COLLABORATION TODAY

Since the early days in 2006 the collaboration has grown. Besides DESY and SNS, BNL, ITER and KEK have joined. The use of shared applications differs from installation to installation. Many CSS instances are running outside the CSS collaboration. There is no need to become a core developer in order to actually run CSS.

SO WHAT IS CSS?

CSS can be configured in many different was. At DESY we decided to make things as simple as possible. So we create just one type of CSS product. Where product means the CSS core and a rich set of plugins. All of these files are zipped to a single zip file which can be unpacked on the end user's machine. In addition to these complete packages it is also possible to just update you CSS instance. The Eclipse update mechanism allows this kind of incremental update when an update site is set up. At DESY this update site is populated with the latest versions of individual plugins. So only this selected plugin gets its update while the rest of CSS stays the same. The core installation with all its settings and configuration files will not be altered.

Even though we are only providing a single kind of CSS product, we have individual user groups which have different views on CSS and are using completely different sets of plugins for their work. This is possible because a role based authentication scheme is implemented in CSS. Logging in to CSS with Kerberos authentication will make sure that only authorized people may perform certain actions on the control system or make changes in the configuration. What is CSS …

… for the Operator

The main plugins for the operator are the tools he needs to control the equipment and to get information when something fails. In addition he wants to look back in history what happened at a certain time or before an event occurred.

The typical tools are: The synoptic display SDS (Synoptic Display Studio), the alarm toolset consisting of the alarm table, alarm tree, message tree and archive message table and the trend browser.

… for the Engineer

The engineer actually is using CSS to configure the EPICS databases using the database creation tool DCT. The EPICS records need proper addresses to read and write from the distributed I/O system. The I/O is configured in the I/O configurator. Sequence programs are running in the front end controllers (IOCs – Input Output Controller). They are actually programmed using the state notation language (snl) editor.

All of these programs are plugins to the CSS toolkit.

… for the Developer

The developer is using the Eclipse Java IDE to create CSS which is based on Eclipse core technology. So – it is a toolkit to generate control system applications. Together with the other collaborators CSS is also a collaboration.

… for the Manager

The manager finds a stable basis he can rely on. New applications can be added easily to the CSS core. After a steep learning curve to create the first plugin it pays off in the following applications to come. A homogeneous look and feel ensures that new plugins will be easily adopted be the end user. There is no different behaviour of each implementation.

Last not least the collaboration has already reasonable support from industry. New developments can also be carried out by industrial partners. The latter is a successful practise at DESY.

SUMMARY

The CSS idea has found its way through the world of (mostly EPICS) control system installations. The community is slowly growing. The collaborators have found a practical non-bureaucratic way to define and implement incremental improvements in the CSS core while keeping as much freedom as possible on the individual applications.

We can be proud of this success!

ACKNOWLEDGMENT

CSS as it stands would not exist without the continuous effort of the collaboration to make it a better product in their day to day developments.

Thanks to the all of the collaborators!

ISBN 978-3-95450-124-3