AGILE DEVELOPMENT AND DEPENDENCY MANAGEMENT FOR INDUSTRIAL CONTROL SYSTEMS

B. Copy, M. Mettälä, CERN, Geneva, Switzerland

Abstract

The production and exploitation of industrial control systems differ substantially from traditional information systems; this is in part due to constraints on the availability and change life-cycle of production systems, as well as their reliance on proprietary protocols and software packages with little support for open development standards [1]. The application of agile software development methods therefore represents a challenge which requires the adoption of existing change and build management tools and approaches that can help bridging the gap and reap the benefits of managed development when dealing with industrial control systems. This paper will consider how agile development tools such as Apache Maven for build management, Hudson for continuous integration or Sonatype Nexus for the operation of "definite media libraries" were leveraged to manage the development life-cyle of the CERN UAB framework [2], as well as other crucial building blocks of the CERN accelerator infrastructure, such as the CERN Common Middleware or the FESA project.

INTRODUCTION

Agile software development methodology characterizes any development method that emphasizes incremental deliveries, working software and fast response to change. It emerged in the 1990s as a response to so-called heavyweight methods, relying on long running project plans, waterfall models and micromanagement.

Agile methods such as SCRUM or Extreme Programming rely on the old maxim “a problem shared is a problem halved”, by encouraging the collaboration of small but cohesive development teams performing fast evolving deliveries. Teams collaborating on a large project will therefore continuously contribute project deliverables to a “common pot”, from which a best-of-breed product release can be composed at any time.

While agile methods emphasize the importance of human interactions over automated processes and customer collaboration over contract negotiations, it is not surprising that software tooling quickly emerged to support operating in such dynamic and fast-paced environments.

Agile Tooling

It would be misleading to consider that agile development can be effective solely with good will, highly motivated independent teams and supporting customers and stakeholders. Agile software tooling aims at freeing software developers from repetitive tasks, while providing immediate and up-to-the-minute feedback on the status of their team's efforts. Developers can therefore confidently test software changes, share their latest progress with their fellow team members without any manual intervention and in the end focus on providing fast updates and concentrating on fulfilling customer requirements.

Agile tooling is necessarily based on software development processes that in most part, originated with heavyweight project management approaches.

SOFTWARE DEVELOPMENT PROCESSES

In order for agile teams to collaborate in a streamlined and manageable manner, well-known project management processes such as :

- Issue management
- Change management
- Dependency management
- Release management

must constantly collaborate and exchange information.

A short summary of each will help us understand their importance, and provide examples of software packages used at CERN to fulfil such demands.

Issue Management

This process focuses on collecting, prioritizing and refining customer demands and internal product quality feedback. Whether use cases, requests for new features or defects identified in existing products, such inputs must be classified and scheduled so as to reduce the risk of unwanted side effects and ensure timely release delivery. Change brought to a product must always take for reference a prior request for such change (once again, to reduce the risk of unidentified and unwanted modifications that could break a working product).

Change Management

This process ensures that changes can be audited, and grouped into coherent set of modifications. Sets of modifications will eventually compose a release. Change management is often seen as a burden by software developers, but becomes a key activity when it is coupled with release management – for instance, once a product version is out in the wild, it is essential to know exactly what differentiates this version from the one that worked better a few weeks or months ago.

Dependency Management

As agile teams happily release ever improving deliverables, being able to orchestrate dependencies among these deliverables becomes an increasingly difficult task. Establishing clear policies and allowing the definition of dependency ranges (*e.g.* by specifying that

Project A can tolerate any of the latest version of Project B, starting from version 1.5 up to version 2.0 and anything in between) becomes a very important aspect of software project management. Considering that software project increasingly rely on third-party open source dependencies, it also becomes important to state and monitor the provenance of such dependencies (*i.e.* if two versions of the same library are available on the internet, one from an untrusted source and one from the official provider of the library, it is naturally preferable to obtain it from the official provider).

Release Management

In order to facilitate software reuse (one of the cornerstones of agile development), software artifacts must be made available in a readily consumable form, so that no time needs to be spent rebuilding such artifacts using the original source (which, if change and dependency management are kept consistent through an established release policy, should of course always be possible). In ITIL v3 [3], a library of best practices in software infrastructure management, a repository used to store readily consumable artifacts is referred to as a **Definite Media Library** (DML). A DML acts as a reference for all artifacts maintained by an organization, and supports features such a search indexes, structured navigation and queries, metadata management and access rights policies enforcement.

Release management also becomes relevant in order to deliver automatic updates to software end-users. Auto-update capabilities have become a common feature of modern operating systems and software packages aimed at the general public.

As an example, the UAB toolset [5] (a development suite for the generation of industrial control systems) offers a Bootstrap utility, illustrated in Fig. 1, which upon startup verifies whether any new versions of the UAB components (such as new device type libraries, new generation plugins etc...) have become available in a central repository.

If new UAB components have been released, end users are offered the possibility to download these components and use them for the generation of their application.

Sonatype Nexus, a commercial DML implementation that acts as an Eclipse P2 repository and a Maven repository, offers a programmatic API (named Aether) which allows to query and download repository contents. The UAB Bootstrap component for instance relies on the Aether API to determine whether new UAB components (in effect, Maven artifacts deployed in Nexus) are available and to download them.

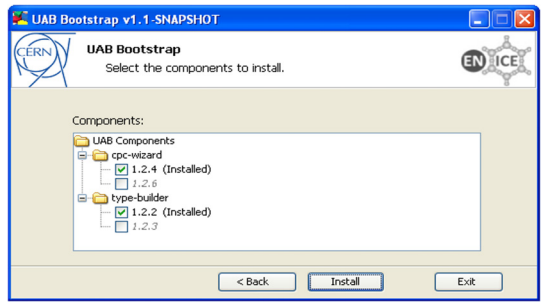

Figure 1: The UAB Bootstrap (using the Aether API).

Continuous Integration

Continuous integration is a quality process that provides constant feedback on changes performed by developers. Coupled with change management, it allows at any point to ensure that developers are informed of regressions (or build breakage) they might introduce in their project.

PROCESSES INTEGRATION

In the context of the UAB project and the LHC Accelerator complex, CERN employs a variety of software packages that support the previously enumerated processes, including :

- **Atlassian JIRA** for issue management
- **Subversion** (SVN) for change management
- **Apache Maven** build for dependency management
- **Sonatype Nexus** as a *definite media library* and release repository for Maven and Eclipse dependencies.
- **Hudson / Atlassian Bamboo** for continuous integration

This best-of-breed selection mixing open-source software with commercial tools operating under various licenses poses necessarily some integration issues. Navigating from one information source is obviously quite possible, as each tool provides navigation links of some sort to its peer processes – but happens to be quite cumbersome when investigating regression issues, plain bugs or releases of insufficient quality due to failure in change or issue management processes.

The CSAR project therefore defines a data format and offers tools that greatly simplify data extraction, navigation and visualization of engineering process related information (when investigating the cause for the introduction of a regression or the presence of a new bug).

Resource Description Framework (RDF)

RDF is a meta-model introduced by the W3C to unify data and meta-data in web-based documents. RDF relies on subject-object-predicate expressions to embed meta-meaning in ordinary data, allowing the data to be processed and leverage by a machine.

RDF acts as an ideal *lingua franca* between our various engineering process support tools, allowing us to harvest all our information and collate it in a navigable way.

Data Visualizations and the Exhibit Framework

Data visualizations represent large amounts of information in an immediately understandable form. Maps, pie charts or tables are traditional visualizations, but others such as heat maps or sparklines can also deliver great expressiveness. In any case, visualizations are decoupled from the data they represent.

The MIT Exhibit framework offer a generic way to represent, query, filter and visualize RDF data. The CSAR project leverages Exhibit and integrates a new visualization yet not supported by the framework for the representation of dependency trees.

Exhibit Reports Build Integration

The CSAR project provides an Apache Maven site plugin – which seamlessly integrates engineering process information into a web-based Maven documentation site [4].

This is particularly useful for continuous integration software packages that support Maven but would not necessarily understand RDF or be able to handle the Exhibit framework.

Fig. 2 presents an example of a Maven generated documentation site integrating an Exhibit report. Once RDF data is provided to Exhibit, the framework automatically generates the required user interfaces elements such as filters and navigation links.

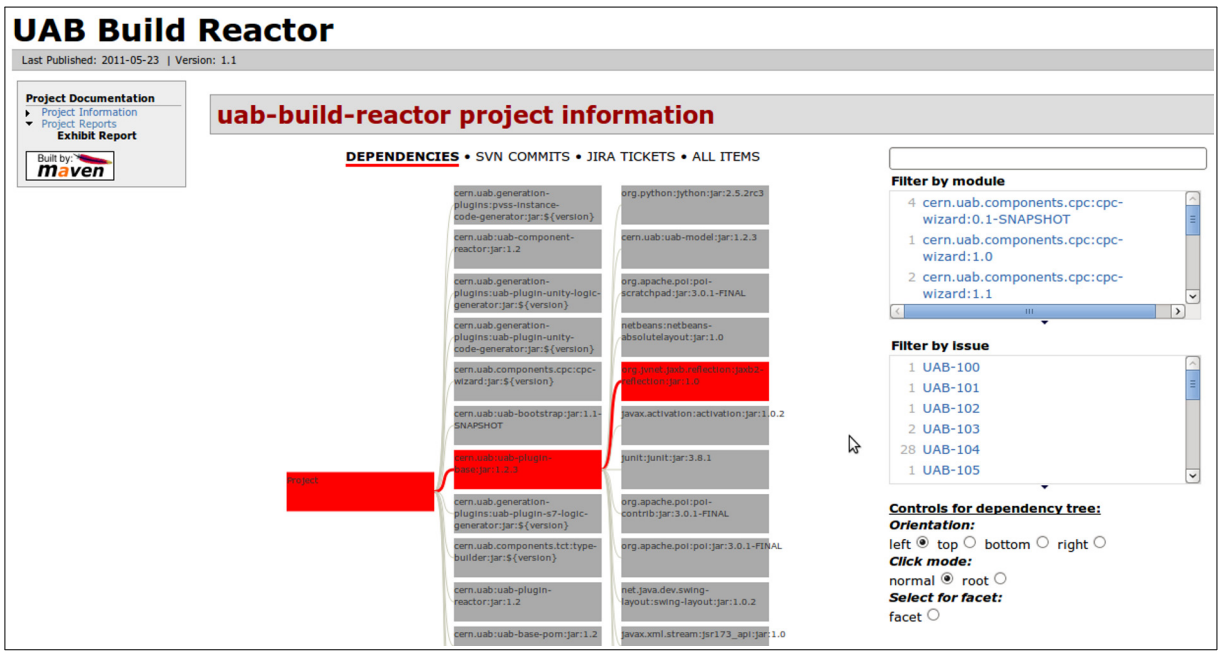

Figure 2: Web-based Maven-generated documentation site with interactive dependency tree and filter fields.

The Exhibit framework also introduces new visualization concepts such as the Timeline visualization, a Javascript widget that allows to browse time-based events, making it particularly suitable for reviewing SVN commit activity or JIRA activity.

Fig. 3 and 4 provide examples of Timeline visualizations as generated by Exhibit according to the RDF data collected by the CSAR collection plugins.

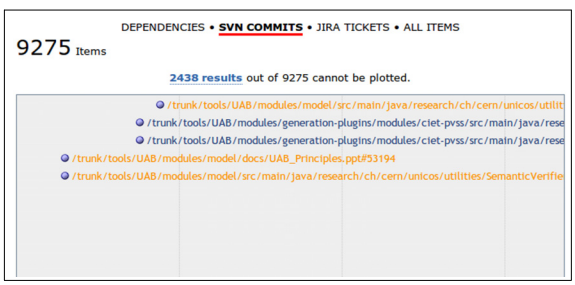

Figure 3: Timeline visualization of SVN activity.

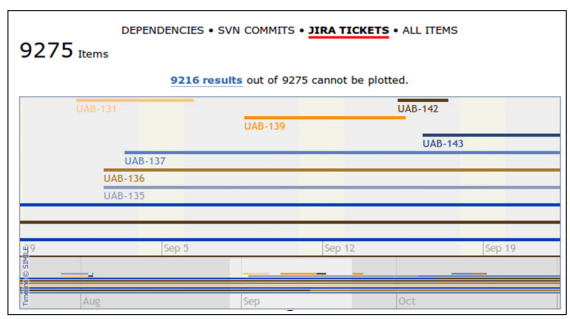

Figure 4: Timeline visualization of JIRA activity.

Timeline visualizations in the Exhibit framework provide navigation links and popup windows which are automatically derived from RDF data.

DATA INTEGRATION

The CSAR project provides Java executables that can be integrated as part of a build process (using their respective Maven plugin wrappers) and extract RDF

information from our various engineering process support systems (JIRA, Maven metadata, Nexus, Subversion).

Once in a unified RDF format, data integration is a matter of collating files together and transforming them to web-friendly RDF dialects (known as Exhibit JSON).

The generation of Exhibit user interface and navigation links is then simply a matter of configuration.

CONCLUSIONS

Integrating software engineering process data sources is useful in order to obtain a global vision of development and release activities. While most engineering process deliver access to structured data, integrating and correlating this information is currently a non-trivial task.

While our usage of RDF as a common data format has certainly proven a workable approach, certain limitations subsist in the current support of RDF. The Exhibit framework for instance was designed highly interactve representation of low volumes of information. A sample extraction of the UAB project activity over the course of 14 months, totalling about 600K lines of code, yielded a 5 megabytes Exhibit JSON input file containing 9275 records, which the Exhibit framework handles with difficulty (requiring a 20 seconds initial startup time and delivering occasional sluggish data filtering performances).

A complete rewrite of the Exhibit framework (Exhibit v3) is interestingly under way in order to ease the integration of third party visualizations and make it able to cope with datasets gathering more than 5 millions of records.

REFERENCES

[1] R. Barillère, Ph. Gayet, *"UNICOS A Framework to build industry-like control systems, principles and methodology ",* ICALEPCS 2005, Geneva, Switzerland, WE2.2-6I

[2] M. Dutour, "Software factory techniques applied to Process Control at CERN", ICALEPCS 2007, Knoxville Tennessee, USA, http://www.JACoW.org.

[3] APM Group, "*What is ITIL ?*", 2007-2011, http://www.itil-officialsite.com/AboutITIL/WhatisITIL.aspx

[4] Apache Maven Project, "*Guide to creating a documentation site*", http://maven.apache.org/guides/mini/guide-site.html

[5] B. Copy et al., "Model Oriented Application Generation for Industrial Control Systems", ICALEPCS 2011, Grenoble, France, WEAAULT02

[6] Massachusetts Institute of Technology (MIT), "*The Exhibit Framework*", 2011, http://www.simile-widgets.org/exhibit3/