FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

On-Process Verification and Report

Tiago Nunes

Project Report

Master in Informatics and Computing Engineering

Supervisor: João Pascoal Faria (PhD)

3rd March, 2009

On-Process Verification and Report

Tiago Nunes

Project Report
Master in Informatics and Computing Engineering

Approved in oral examination by the committee:

Chair: Ana Cristina Ramada Paiva Pimenta (PhD)

External Examiner: Fernando Brito e Abreu (PhD)
Internal Examiner: João Carlos Pascoal de Faria (PhD)

19^{th} March, 2009

Abstract

This report describes the motivation, architecture and post-implementation return on investment of an On-Process Validation and Report solution. Its objective was to create an automation tool that would free the Product Assurance team at Critical Software, S.A. from repetitive and time-consuming tasks, while at the same time providing a platform upon which further systems could be implemented, augmenting the tool kit of the Product Assurance Engineers. This was achieved by designing a modular and extensible platform using cutting-edge Java technologies, and following a goal-oriented development process that focused on iteratively providing as many domain-specific features as possible. It is shown that the resulting system has the potential for considerably reducing costs in the validation of projects throughout their entire life cycle.

Resumo

Este relatório descreve a motivação, arquitectura e o retorno sobre o investimento após a implementação de uma solução para a Validação e Reporting On-Process. O seu objectivo era a criação de uma ferramenta de automação que libertaria a equipa de Controlo de Produto da Critical Software, S.A. de tarefas repetitivas e demoradas, e ao mesmo tempo fornecer uma plataforma sobre a qual futuros sistemas poderiam ser implementados, aumentando o leque de ferramentas dos Engenheiros de Controlo de Produto. Isto foi atingido desenhando uma plataforma modular e extensível usando tecnologias Java de vanguarda, e seguindo um processo de desenvolvimento orientado aos objectivos que se concentrou em fornecer iterativamente o maior número possível de funcionalidades específicas ao domínio. Mostra-se que o sistema resultante tem potencial para reduzir consideravelmente os custos da validação de projectos ao longo de todo o seu ciclo de vida.

Acknowledgements

Very rarely is a big endeavour the work of a single man. By providing me with support of all kinds, from technical to moral, several people helped making this project possible.

Joel Oliveira, who assumed and balanced (juggled?) the roles of client and mentor for the project, was an inspiring leader in my transition from the academic to the professional world (or as he would often call it, the *jungle*). Braselina Sousa, my tutor at Critical Software and project manager, was always very methodical and restless in making sure I had everything I needed to do a good job. Professor Pascoal Faria, my supervisor at FEUP, was always extremely reliable, and his experience in the field proved invaluable when providing feedback throughout the meetings and the development of this report. A special thank you to these people who helped me navigate the project towards success.

All work and no play makes Jack a dull boy, and I was blessed to be involved in a friendly and fun environment from my very first day at Critical Software. I would like to thank Braselina Sousa, Joana Gonçalves, José Pacheco, Maurício Costa and Nuno Salvaterra for keeping me mentally sane during these months - I never thought coffee breaks could be so much fun.

Finally, my very special thanks to my family, for every reason. Never have I been short of absolutely anything, neither have I been a spoiled boy, and this is obviously a fruit of a *tremendous* amount of your hard, good and fair work throughout your lives. Thank you.

Tiago Nunes

"That's thirty minutes away.
I'll be there in ten."

Winston Wolf, in *Pulp Fiction*

Contents

List of Figures

LIST OF FIGURES

Abbreviations

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CSW	Critical Software, S.A.
CVS	Concurrent Versions System
FEUP	Faculdade de Engenharia da Universidade do Porto
KOM	Kick-Off Meeting
PCM	Project Close Meeting
PM	Project Manager
QMS	Quality Management System
RAP	Rich AJAX Platform
RCP	Rich Client Platform
RWT	RAP Widget Toolkit
SPA	Software Product Assurance
SPACE	Name of the tool developed within the project (not an abbreviation)
SPAE	Software Product Assurance Engineer
SPAM	Software Product Assurance Manager
SWT	Standard Widget Toolkit

Chapter 1

Introduction

The goal of this chapter is to introduce the context of the report. We will briefly describe the project, its stakeholders and objectives, and define the structure of this document.

1.1 Context

The project reported by this document was developed within the frame of the Project unit of the Master in Informatics and Computing Engineering, at FEUP, and in cooperation with Critical Software, S.A. It was designed to provide the student undertaking the task with a direct experience in the professional environment while applying the concepts acquired throughout the University degree.

1.2 Critical Software

Critical Software, S.A. (hereinafter referred to as Critical) provides solutions, services and technologies for mission and business critical information systems across several markets. Since its foundation in 1998 and having closed contracts with major worldwide companies and institutions such as NASA, ESA, United Kingdom's Ministry of Defence and AugustaWestland [Cri09], Critical has maintained a yearly double-figure growth in turnover and has been awarded consecutive honours for its successes.

One of the main focuses at Critical is software quality, as it is a strong pre-requisite in the mature markets in which the company operates. The software development process within the company is continuously improving, and its QMS has been certified to high demanding standards, such as CMMI Level3 (being the first Portuguese company to obtain this prestigious quality certification), ISO 9001:2000 Tick-IT (this time the first in the Iberian peninsula).

1.3 Motivation and Objectives

Following such high concern towards the produced software's quality, projects developed at Critical must abide to the processes, procedures, policies and templates defined in the QMS.

This project is expected to result in an automation of the constant and independent verification and evaluation of the projects throughout their entire life cycle, assessing whether these are following the QMS (On-Process) or not, as well as to lay the foundation not only for future improvements in the system but also for new additions to the verification tool kit. It contemplates the requirements analysis and architecture of a web-based solution for this problem, as well as the implementation and validation of those requirements.

1.4 Structure

Apart from this introduction, this report contains four more chapters.

In chapter 2, we will define the project and its goals. The methodologies and plan will also be described in more detail, after describing the state of the art concerning the On-Process validation at Critical before this project was developed.

In chapter 3, we will focus on the project's design and implementation. The applied technologies will be described, and we will provide a progressively detailed view over the architecture and the decisions that were taken.

In chapter 4, the product developed will be presented, demonstrating the user experience and explaining the return on the investment provided by it. Before, we explain issues that blocked the development of certain features.

In chapter 5, we assess the fulfilment of the goals that were set in the beginning of the project and explain how it lays the foundation for future improvements. The major obstacles found during the project will also be outlined.

Chapter 2

Project Definition

This chapter defines the requirements for SPACE, the tool responsible for on-process verification and report to be developed in this project. The state of the art in Software Product Assurance before the inception of the tool will be described, and we will proceed to present its goals, along with the methodology and plan followed throughout its development.

2.1 State of the Art - SPA

The SQA department at Critical Software had been responsible, since the very inception of the company and with little more than twenty employees, for maintaining an effective QMS system. This included establishing processes and practices as well as assessing projects in their conformity.

In 2008, however, the company felt the need to distinguish Quality Assurance, responsible for the definition and improvement of the software development processes, from Product Assurance, which would deal with the assessment of compliance of projects to these processes. Also, the demanding quality certifications Critical relies on to establish itself in the more mature markets, especially CMMI, require that the project teams shall not be involved in the testing of their own developments - testing shall be done by an independent expert team [MBC06].

The Software Product Assurance (SPA) area was therefore created, becoming responsible for independently evaluating projects inside Critical. The company now possesses one SPA team for each area of application (Enterprise Critical Solutions and Business Critical Solutions). Each team is led by the SPA Manager (SPAM), with coordination responsibilities over all the SPA Engineers (SPAEs). The responsibilities of the SPAEs

include the definition of test plans, test execution, and on-process validation and report, i.e checking the project's conformity against the QMS at any moment in its life cycle.

2.1.1 SPA Logbook

The on-process validation was made using a spreadsheet containing the checklists defined by the QMS for each phase of the project and a set of predefined milestones, or checkpoints, at which the verifications would take place. The appendix A lists the verifications currently made by the SPA team. The conformity result for the project would be calculated by the spreadsheet, allowing the creation of a report based on this figure and the verification items worthy of notice.

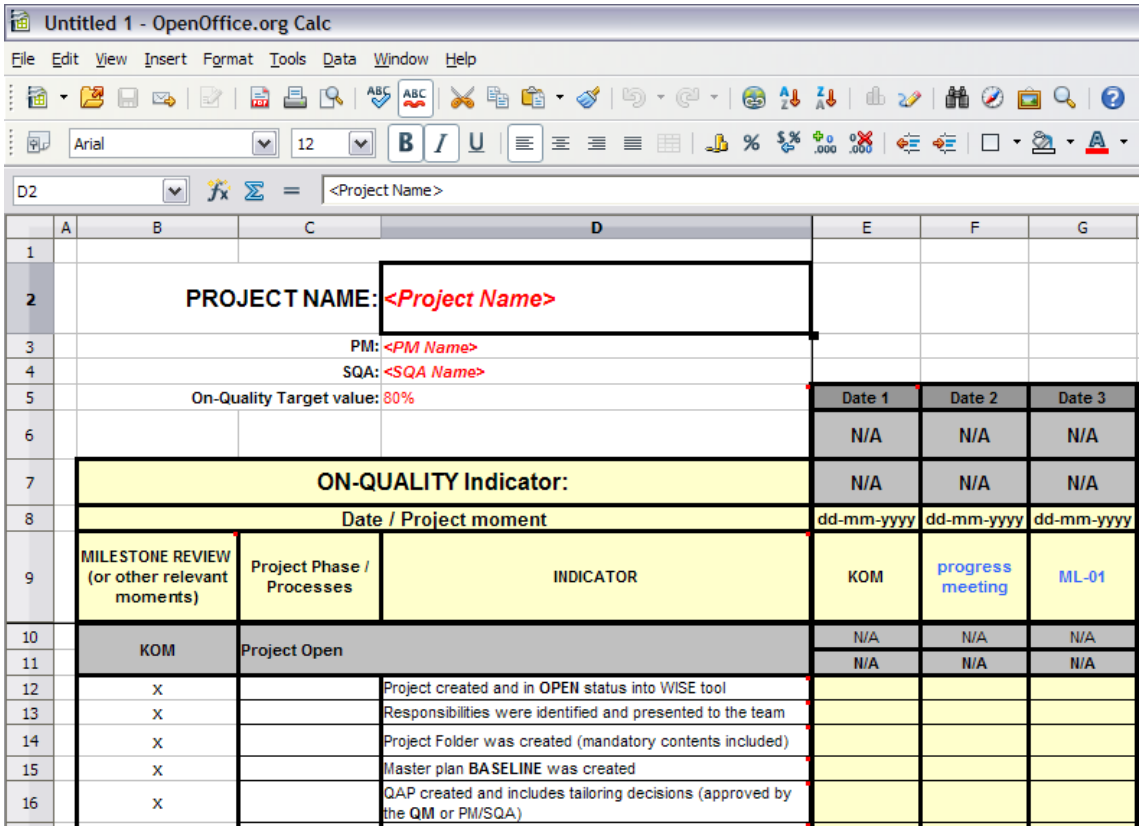

Figure 2.1: SPA Logbook

Each item on the checklist represents a verification to be made. These would mostly include checking the presence of artefacts in the version control system and of information in several information systems. Examples of verifications would be:

- Software Architecture Specification is approved;
- Code conventions are being followed;
- Testing strategy was followed.

For each project, the spreadsheet template was tailored according to the project's needs - unnecessary checks would be hidden and the defined milestones set.

The SPAE, facing the need to perform a validation, would take each item on the checklist one at a time and perform its action. He would then fill in the cell matching the checklist item and the current checkpoint with the result of the validation. In project planning, 10% of the SPAE's effort would usually be allocated to on-process verification and report.

2.2 SPACE

The idea for this project arose from the repetitive nature of the tasks involved in the on-process verification. The checks contained in the SPA Logbook had to be manually validated by referring to the corresponding information source. This literally meant browsing through directory structures checking whether certain artefacts were available, and database-driven dynamic web-pages checking whether certain information was available.

Most of these tasks are serious candidates for being automated by a single software tool, greatly reducing the SPAE's effort in these verifications while at the same time providing further independence of the project team. SPACE was therefore idealized as a checklist automation tool that would allow constant verification and report over development projects throughout their entire life cycle, verifying their conformity to the QMS.

This feature alone was expected to drastically improve the efficiency of the on-process verification. However, while all the features offered by the SPA Logbook were to be ported to the new system, the concept of automation opened doors to further improvements in the verification process.

Should good development practices be employed, concepts used in the construction of this tool would be easily translatable to other domains apart from on-process validation. Therefore, it became an objective to design the tool in a way that would make it as general as possible, taking future adaptations in mind.

Moreover, other projects for the SPA area have been idealized, and would benefit from a framework over which to be deployed. SPACE was finally defined as a framework for the SPA tool kit, in which the on-process validation and report software would be the first tool to be developed.

2.3 Methodology and Plan

The first two weeks were dedicated to training - company procedures and introduction to the project and domain - and the study of technologies. This provided the team with better insight over the effort needed to accomplish the established goals. As such, the requirements engineering phase followed, for a period of four weeks, in which the requirements, architecture and test plans were developed and reviewed.

An iterative approach to development was decided upon, and three development milestones were decided upon: the first after two weeks of development, in which the modules and initial framework would be set up; the second after another five weeks, within which the bulk of the application would be developed, followed by two weeks of validation in parallel with the production of this report; and a final set of five weeks for further developments and the eventual need for fixes brought to light after the validation.

An extra set of four weeks of contract with Critical Software, beyond the twenty weeks expected by the Project unit, will be used as an operations and maintenance phase during which the tool will be used in production and further improved.

2.4 Summary

This chapter presented the motivation for the project reported by this document. We explored how the problem was addressed in the past and how Critical envisioned that it could be improved. We therefore defined SPACE as the solution to this problem, and outlined the plan that was followed in developing the system.

Chapter 3

Architecture and Design

In his chapter we will focus on the project's design and implementation. The applied technologies will be described, and we will provide a progressively detailed view over the architecture and the decisions that were taken.

3.1 Actors

The system features only one actor, the SPAE, who is responsible for tailoring the checklist according to its reality and needs and for editing and validating the results of the automated checklist execution.

The Project Manager (PM) is also involved, by optionally receiving notifications of existing reports. The information gathered from the tool will in any case be communicated to the PM, either directly by the tool or by the SPAE, so that it allows him to take corrective actions on the project if it has been determined that such action is needed.

3.2 Requirements

3.2.1 Business Requirements

The business process model for SPACE is presented in figure 3.1. Three processes are identified:

- Checklist tailoring, in which a user enables or disables checks and checkpoints according to each project's reality;
- Checklist execution, which may happen by user request of by the triggering of a scheduled event. All the checks enabled for a specific project are executed, by querying specific sources for the information needed. The results of the execution

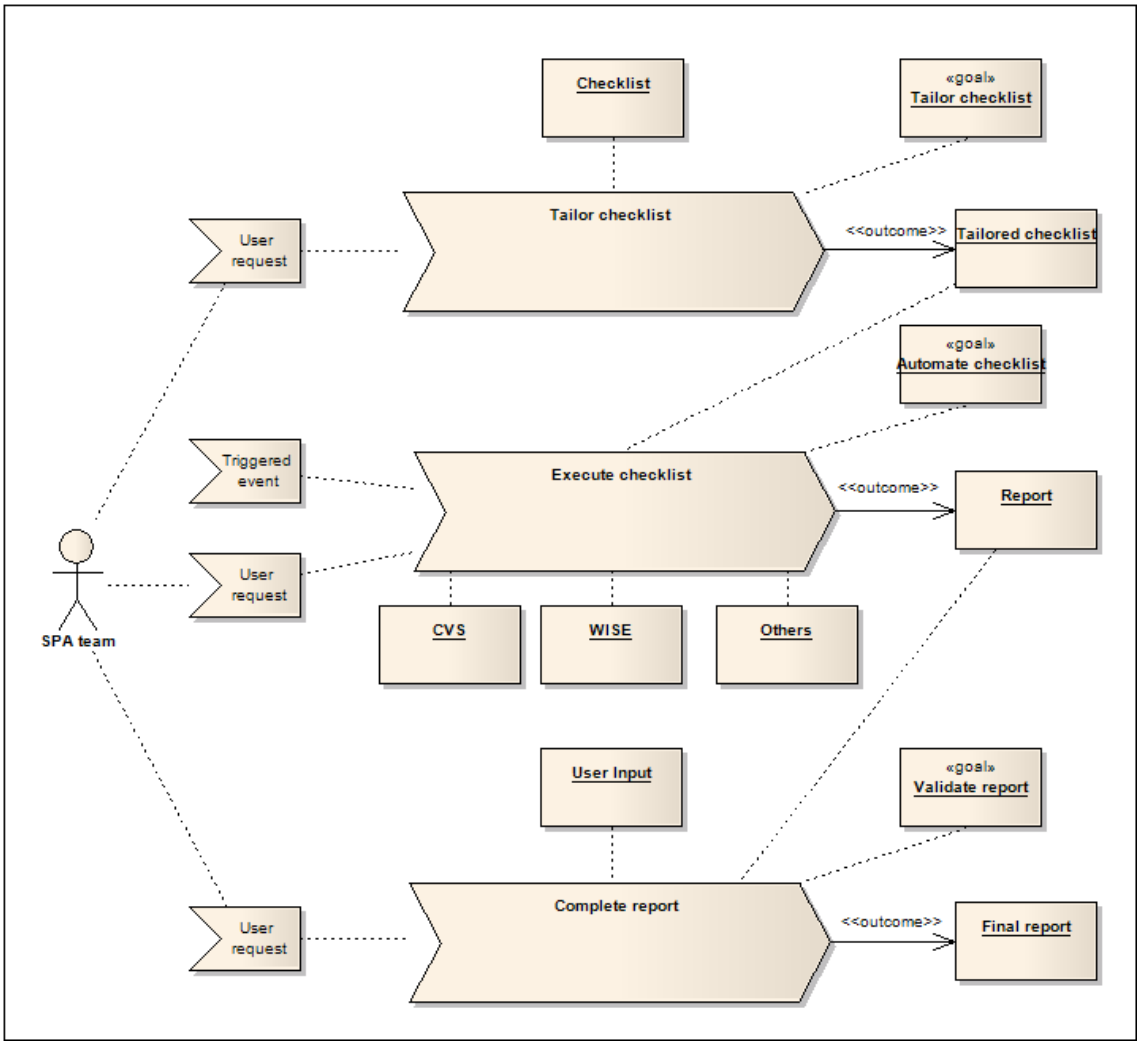

Figure 3.1: Business process model

are recorded into the database for further validation by an expert. Stakeholders (SPAE and PM) may be notified of the event.

- Report validation. The expert user (SPAE) will validate the automation results and complete the report with any information considered relevant.

The tool also provides administration access, where new checks and checkpoints may be added for general use, and system-wide configurations may be made.

3.2.2 Architectural Requirements

SPACE was intended to build upon the existing SPA Logbook spreadsheet. Therefore, it implements the current features of the document and provides further functionality. The migration shall be complete once the final system has been implemented - there shall be no need to revert back to the SPA Logbook.

The SPAEs, possessing a working knowledge of the domain, shall find the transition from the SPA Logbook to the SPACE tool natural and seamless. For this reason, the grid-like relationship between checks and checkpoints was maintained in the final system.

In the long run, it is desirable to adapt the system to fulfil the same needs (checklist tailoring, automation and reporting) in different contexts, such as Quality Assurance. The system provides full flexibility, not having any domain-specific bindings.

The tailoring of a checklist and checkpoints to better suit a project's needs were already contemplated in the SPA Logbook, by means of hiding unnecessary columns or rows in the spreadsheet. The addition of new features (namely new checks), however, was supported only by adding new data to the spreadsheet. This implied that this information had to be published to all the users (SPAEs) by updating the spreadsheet template, teaching them the method for verifying this new item and importing the changes upstream into possibly multiple Logbooks already created for each project. This problem urged a centralisation of the information. This way, it was idealised that the SPACE tool should be used as a remote central service.

Taking into account all the previous considerations, interaction with the SPACE system follows a client-server model, as we can see in figure 3.2. The information is kept in a database accessed and manipulated by the tool. The verification automations will query different information sources in the CSW network (see section 3.4.1 for the identified sources). The user interface will be provided as a web application, accessible by a web browser in the client machine.

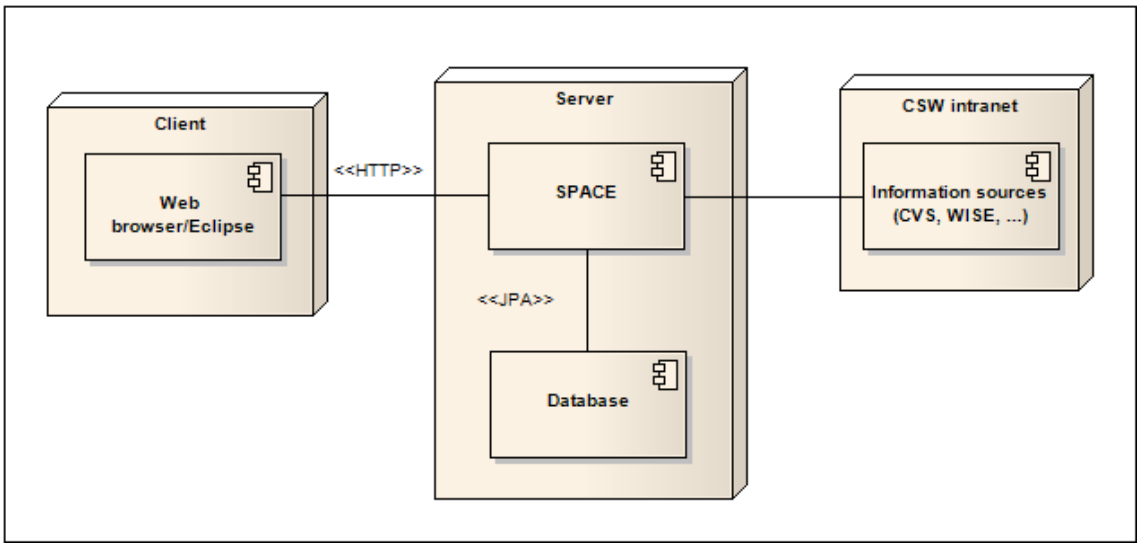

Figure 3.2: Deployment model

One of the client requirements was that the system shall be developed for working on top of an OSGi framework implementation (namely the Equinox implementation). This framework supports modular designs [OSG07a] and provides the architecture with

services that simplify several issues such as installation, update and interfacing between modules (bundles) [OSG07b].

As specified in section 3.4.1 (System Decomposition) the architecture takes advantage of this model, separating the business core module from plug-in modules that are more prone to change, either by the need for frequent updates, completely different approaches (such as another interface apart from the web page) or even a context change (for example adapting the tool to validate a checklist for a different application domain, demanding an entirely new automation module). The core module is able to function without the need for any of the specified plug-ins - obviously with corresponding limitations.

The major architectural issue has been the Check entity. It combines both a database side with relevant information about its use in different checkpoints/projects/scopes and a programmatic side that defines the automation that is performed to validate the check.

It is desirable that users may create new checks as the necessity arises. However, the basic creation of the check shall only generate the database side of the entity, while the programmatic side will be implemented and plugged-into the system by the system maintainer after a request from the user. Note that this request may never happen, in the cases where an automation of the verification is not necessary or even possible. That specific case is signalled accordingly by the tool.

This led to the design detailed in the following section, in which the automation bundle is completely independent from the core system, and in fact may not even exist - in this case the tool works merely as a centralised SPA Logbook, needing an expert user to manually perform the validations and insert the results into the database.

3.3 Technologies

Several technologies were deployed in the development of this project. In this section we will list the most relevant in terms of direct influence in the design of the solution.

This set of technologies had been decided upon beforehand by the company. Although this eliminated the need for research concerning alternatives, we will present them and the added value they brought to the development of the project.

3.3.1 OSGi

The OSGi Alliance, formed in 1999 by several market players such as IBM, Nokia Corporation, Oracle Corporation and Siemens AG [OSG09a], is an independent non-profit corporation developing a framework for interoperability of applications. This framework, OSGi (which formerly stood for Open Services Gateway initiative), is intended to fix the Java technology's modularity shortcomings by implementing a dynamic module system framework that applications can build upon.

Essentially a service-oriented plug-in system, the OSGi Service Platform allows the dynamic loading of new functionality into deployed systems while also managing the visibility, life cycle, dependencies and versions of the plug-ins (referred to as *bundles* in the domain).

> "OSGi technology adopters benefit from improved time-to-market and reduced development costs because OSGi technology provides for the integration of pre-built and pre-tested component subsystems. The technology also reduces maintenance costs and enables unique new aftermarket opportunities because components can be dynamically delivered to devices in the field." [OSG09b]

3.3.1.1 Equinox

There are numerous implementations of the OSGi R4 framework specifications. Although several of these are commercial, there is also a number of freely available options like Apache's Felix, Knoplerfish and the one used in this project, Equinox.

Equinox, an Eclipse project (and the base for all Eclipse since 2003) [Ecl09e], is currently the only freely available implementation that has been certified by the OSGi Compliance and Certification program, assuring interoperability for applications and services running on its platform. The readiness of the Eclipse Platform in what concerns OSGi development adds further value in choosing this implementation.

3.3.2 RAP

Rich AJAX Platform is a fairly recent addition to the Eclipse Project, with its first version released in the end of 2007 [Ecl09b]. Its aim is to take another Eclipse project, RCP (Rich Client Platform) [JM05], to the realm of web applications, providing abstraction both to the developer over the implementation and to the user over the utilisation of such applications.

> "The RAP project enables developers to build rich, Ajax-enabled Web applications by using the Eclipse development model, plug-ins with the well known Eclipse workbench extension points, JFace, and a widget toolkit with SWT API." [Ecl09b]

In the developer's perspective, the entire Java API is available for the implementation on the server side. The presentation on the client side, a "rich web page" accessed through any modern web browser, is taken care of by *qooxdoo*, an AJAX application framework [Qoo09]. The developer is therefore free from having to understand HTML,

CSS or DOM, let alone to worry about writing interfaces that support all the major web browsers.

Almost exactly the same source code may be used both for developing rich (thick) client applications and rich AJAX applications - these small differences, related to the extra needs of the client-server architectural model, are isolated in and abstracted by the RWT (RAP Widget Toolkit) layer, which implements the same API as RCP's SWT (Standard Widget Toolkit). This which allows for the knowledge acquired in developing for RAP to be entirely usable for developing for RCP, and vice versa.

In the user's perspective, all the interaction with the system is performed through a Javascript-enabled browser. The look and feel of the local operating system's native interfaces is maintained, although branding of the products may be performed to customize the user experience. This grants familiarity to the user, who will easily recognize all of the controls he already interacts with on a daily basis. Figure 3.3 is an example of a RAP application interface.

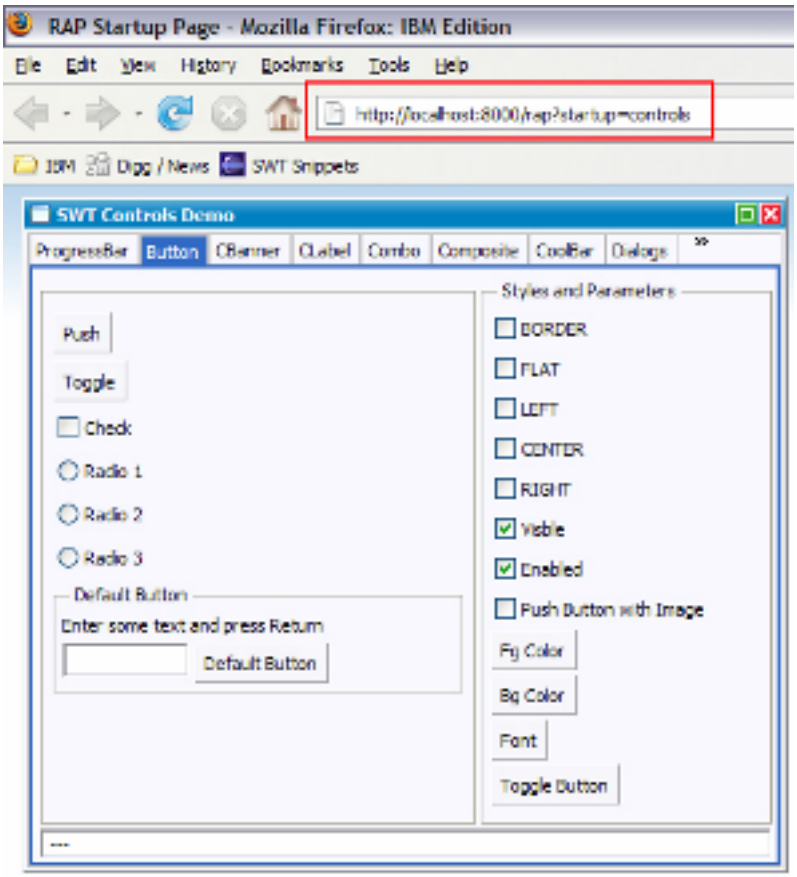

Figure 3.3: Example of a RAP application [IBM09]

Currently under active development, the RAP project is becoming a solid option for developing rich centralized applications with high demands for server-side logic. As

pointed out in section 5.2 (Major Issues), however, the project has not yet reached maturity, which especially in the first months of implementation resulted in slowdowns in development. Later releases were increasingly feature-complete [Ecl09c].

3.3.3 EclipseLink

The project at hand had requirements that demanded the use of a database for persisting relevant information. An approach through object/relational mapping was chosen, adding this layer between the application and the database, improving the software's robustness by requiring less error-prone connection code and also making it more portable and therefore tolerant to database changes.

The Java Persistence API (JPA) is the Java Platform's framework for managing relational data within applications [Mic09], enabling objects to be persisted to a relational database by use of object/relational metadata. EclipseLink [Ecl09d], originally contributed by Oracle's source code release from its TopLink product in 2006, has become the reference implementation for JPA 2.0 [Ecl09a]. Aside from providing support for JPA it adds further flexibility by supporting other standard persistence related APIs (such as Java XML Binding and Service Data Objects), securing several options for adapting to future requirements.

> "The Eclipse Persistence Services Project (EclipseLink) project's goal is to provide a complete persistence framework that is both comprehensive and universal. It will run in any Java environment and read and write objects to virtually any type of data source, including relational databases, XML, or EIS systems. EclipseLink will focus on providing leading edge support, including advanced feature extensions, for the dominant persistence standards for each target data source" [Ecl09d]

3.4 Architecture

In this section we will detail the architectural decisions that were taken while designing SPACE. Building on the requirements and technologies elicited above, the provided vision will include the system decomposition and a detail of the data structures.

3.4.1 System Decomposition

Figure 3.4 details the modular approach chosen for the development of SPACE.

As described in the Architectural Requirements, the system is comprised of a core module and functionality plug-ins. The first is independent of the others, although it provides very little functionality by itself. The goal is to take advantage of the OSGi

framework underneath and so to allow the main system to run while being able to hot-plug new functionalities.

The OSGi framework provides a Service Registry [OSG07b], in which modules (bundles) register the services they provide and look for the services they consume. All the logic dealing with the (de)activation of bundles is of the framework's responsibility. The interfaces specified in the component module above will make use of this system, greatly simplifying development while at the same time making the system considerably loose-coupled and therefore tolerant to change, as desired.

The tool is divided in the following components:

- SPACE core: this component provides the work flow of the system. It contains the data structures necessary to execution, and relies on plug-ins to populate them with data. It also queries basic project information from WISE, Critical's information system holding all project related data. Additionally, it contains a proxy [EG94] for the Main Automation plug-in (detailed below), that serves as a void place holder for the automation of a check while it is not available, and is responsible for delegating execution to it when it is.

- Main Automation: this component aggregates all the different check automations. New automations will imply a new version of the component, which will be easily replaced thanks to the underlying framework. The automations will require information from external systems (see below).

- RAP Interface: the interface layer was developed with the Eclipse Rich Ajax Platform technology, as a client requirement. The core component will specify interfaces for this layer to access and manipulate the data structures.

- Database and Database Access: the core component will provide an abstract interface to populate the data structures that allow future changes to the way the data is stored. This was achieved using the JPA object/relational mapping implementation of EclipseLink. Meanwhile, the data will be kept in a Database, and the Database Access component works as a wrapper of the Database API to the SPACE core interface specification.

SPACE interacts with several external tools for data input:

- CVS: Location of documents (artefacts, inputs, outputs);

- WISE: Project info (status, plan, baseline, milestones, lessons learned, etc.);

- DWIN: Traceability (User requirements-Software requirements, reqs.-architecture, reqs.-tests, etc.);

- QMETRICS: Quality metrics;
- JIRA: Issue tracker;
- WOW: Identification of open work orders.

3.4.2 Database Design

The data model depicted in figure 3.5, describes the database design devised for SPACE. It is intended to hold not only the records of past executions but also the static data needed by the tool, such as the checks and checkpoint types available, the scopes (hierarchy levels) they are contained in and default weights for checks.

- Checks - holds information on the available checks. The mapping to its programmatic automation is done via its ID. It belongs to a single scope;
- Scopes - lists all the hierarchy levels containing checks. Each scope may have either one parent or no parent at all, and may be the parent of several scopes.
- Checkpoint_types - contains the base types of checkpoints that a project may have. These include all the milestones (from the Kick-Off Meeting (KOM) to the Project Close Meeting (PCM)), progress meetings, etc.
- Default_weights - relates checks with all possible checkpoint types by defining the default minimum weight each check may have when considered during a checkpoint.
- Checkpoints - keeps a record of all the checkpoints enabled for projects. Almost as an instantiation of a Checkpoint_type, it holds instance-specific data, including the date of execution and user preferences, such as the possibility of user notification. Notifications may be scheduled or not.
- Validations - holds information on the preparation and results of the check executions. Relating the checks with the specific checkpoints, it will result from the tailoring of the checklist, may hold a tailored weight as well, and will hold the result of the validation and comments by the expert.

3.5 Summary

This chapter went into detail over how the final solution was designed. The requirements and technologies that have been used to implement them were described alongside the main reasons for their election, and afterwards we explored with increasing detail the architectural decisions that were taken during the design phase of the project.

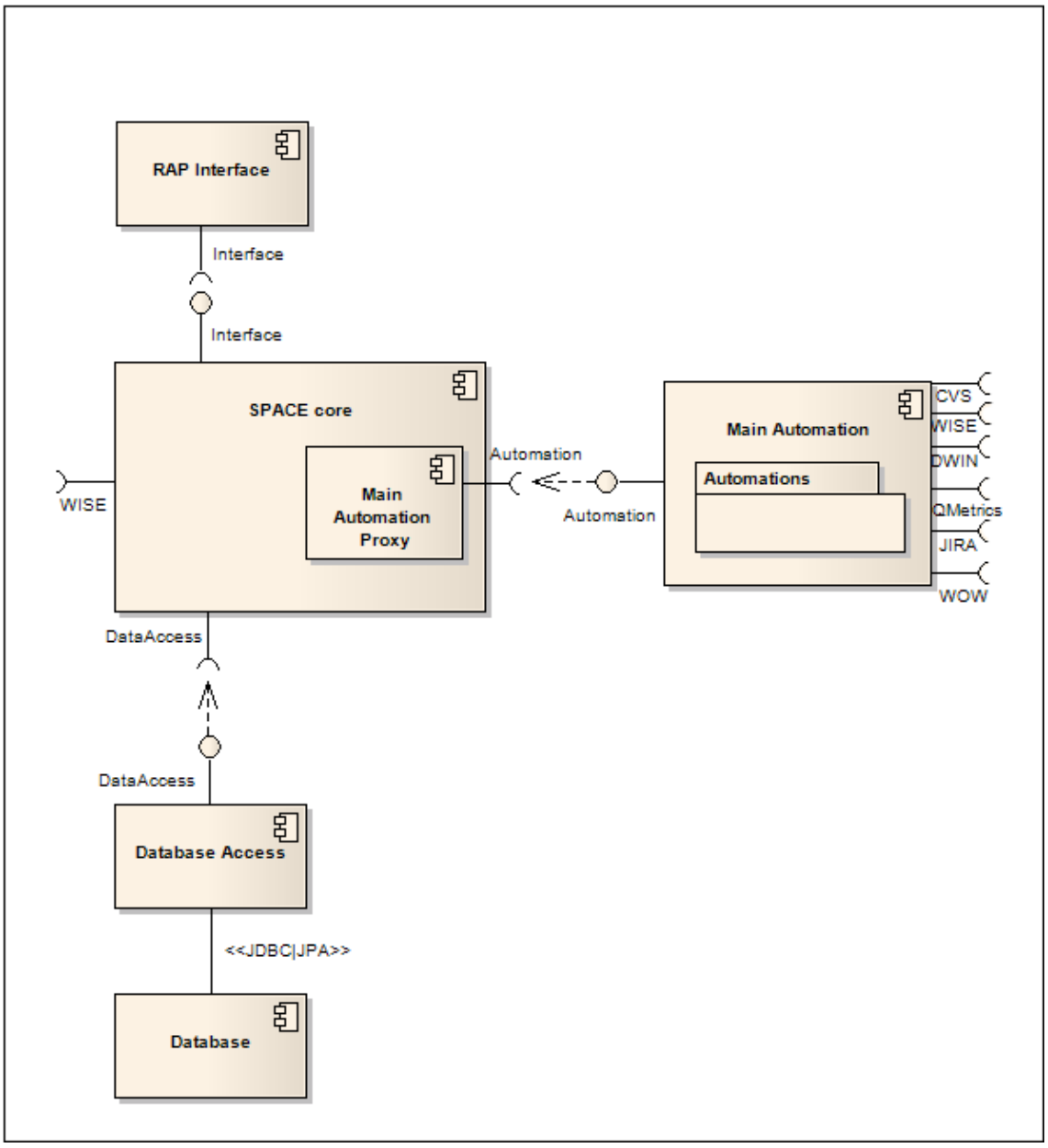

Figure 3.4: Component model

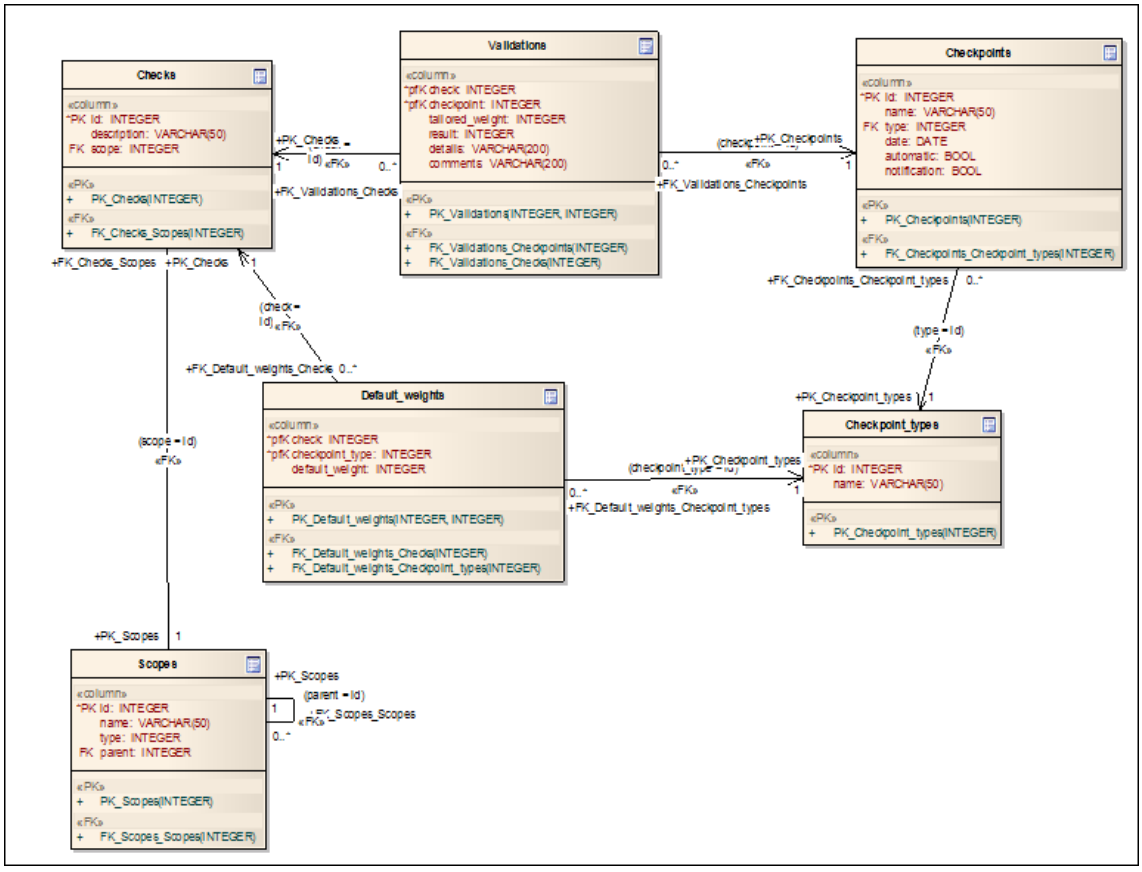

Figure 3.5: Data model

Chapter 4

Product

This chapter will present the product developed according to the specification in the previous chapters. After explaining a set of issues that hindered the development, we provide an overview of the user experience in operating SPACE, and explain the Return On Investment provided by this solution.

4.1 Blocking issues

Before the presentation of the current state of the platform, it is important to mention two critical issues. These issues blocked the development of features to the date of this document, but are expected to be resolved in the near future.

The first problem was the integration with Critical's information system, WISE, from which SPACE is to withdraw important data. This involved a database connection that due to network topology definitions did not work as expected. This issue was reported and was being resolved by the responsible team as of the date of this report.

SPACE was therefore fed with fictitious data during development. While the implementation is already complete in what concerns the manipulation of the data once it is received and parsed accordingly, this parsing will require further development effort once the issue is resolved.

The second issue concerns the reuse of the chart widgets developed at Critical. As mentioned in the Technologies section 3.3, the RAP technology was at the date of this project still under heavy development. This sometimes triggered the breaking of software implemented towards previous versions of the technology when new versions were released.

A previous project at Critical required the development of a widget for generating charts in a RAP application. This widget was signalled as a reuse possibility for SPACE,

which could use it to provide a more intuitive report over the on-process validation and report history throughout the projects' life cycle. During the first two weeks of the internship in early October 2008, dedicated to training and technology studying, several small applications were developed as proof-of-concept for technology features to be implemented in the final project. One of these small applications made use of the provided widget and demonstrated the relevant functionalities. It was implemented against RAP version 1.1.1 Service Release, released in September 2008 [Ecl09c].

In February 2009, the implementation of the reporting module for SPACE began, and the widget integration was not as straightforward as expected. RAP had reached version 1.2 Milestone 5, released in February 2009, with several and major improvements throughout the frequent (monthly) releases. On the other hand, the chart widget did not work. An update was solicited to the team that developed the widget, and the issue was being resolved at the date of this report.

Taking into account the work developed in the project's first two weeks, where the chart widget's API was explored, feeding it with the already existent data (presented in tabular form) shall be straightforward once the issue is resolved.

4.2 User Experience

Figure 4.1 is a screenshot of the tailoring perspective in a possible work session undertaken by a SPAE.

As per the requirements, the SPACE platform is accessed via web using a modern web browser (provided it is supported by qooxdoo, the AJAX framework used by RAP).

In the left view is the project explorer, where the SPAE selects the project he is currently working on from the list of projects he is currently assigned to.

In the main editor window, all the information concerning the project is provided, in the three tabs - Project Information, with details of the project gathered from the company's information system; Tailoring, where an editable table allows for the definition of each validation's weight in each checkpoint; and Report, where the previous table is now presented with the results of the executed validations and a graphical history of past results.

New checkpoints may be added based on already existing checkpoints, to support progress meetings and ad-hoc validations desired by the user. All the checks are available for tailoring, being automatically disabled when not being used in any checkpoint.

In the lower view, details of each check are provided upon selection in the editor, and details of each validation result are provided also upon selection of the desired result.

In the toolbar, the user has the option to run the next batch of validations, which corresponds to the nearest checkpoint in time. This will result in the execution of every

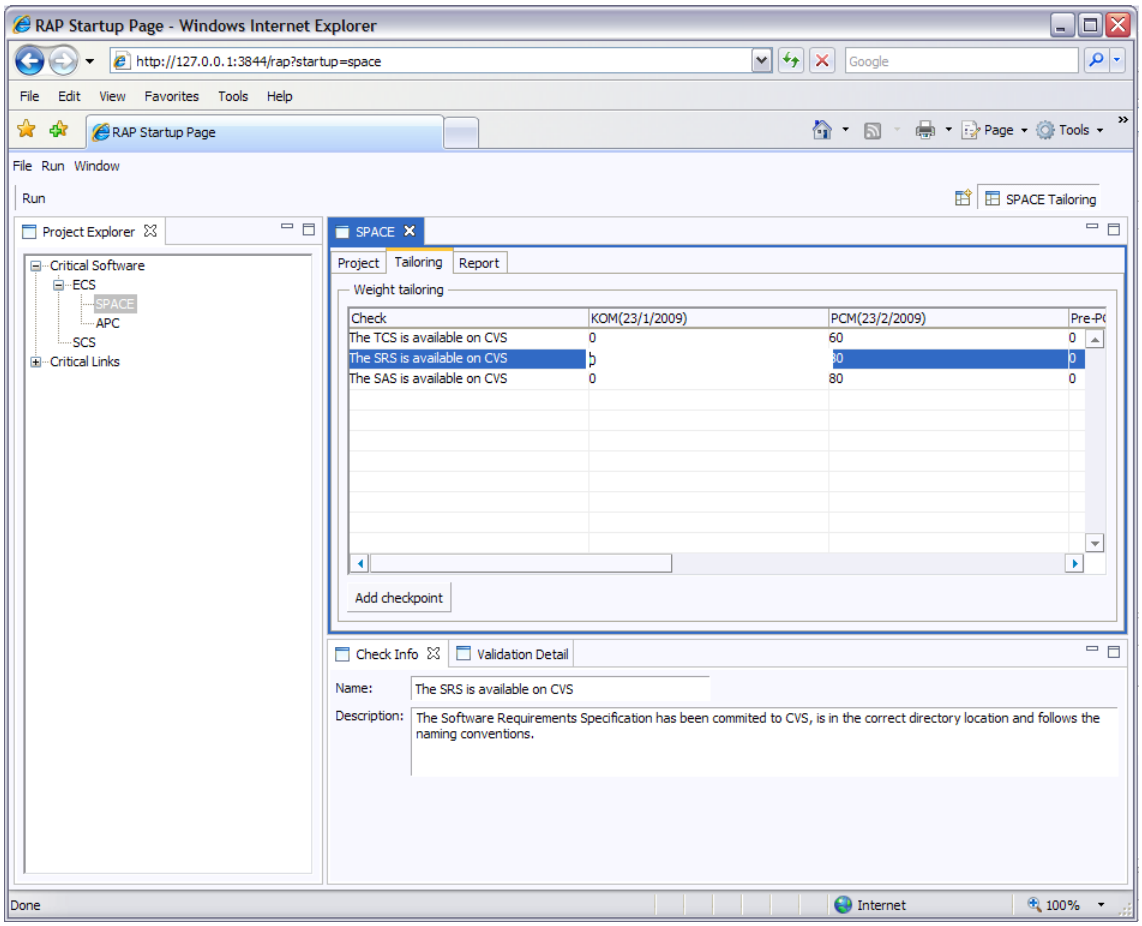

Figure 4.1: SPACE - tailoring perspective

check's automation and the presentation of the execution results in the Report tab, shown in figure 4.2.

One parametrisable automation was developed as a proof of concept and for being used as an example in the development. This is in fact an automation that will be heavily used, as it checks the version control system (in Critical's case and at the time of writing, CVS[1]) for the presence of a certain artefact, as provided in the parameters. The three example validations visible in the figure above make use of this automation to check the CVS for three different documents.

Also in the toolbar the user with the correct privileges may switch to the repository perspective, as shown in figure 4.3, where he may administer the available checks and checkpoints for the system.

The left view is now an explorer over the various domain entities used by the system. It will be a point of confluence for other tools that may be implemented on top of SPACE - they shall contribute to this explorer view with their own entities to be managed. Currently, only the checks and checkpoints are featured.

[1]Concurrent Versions System, a free software revision control system

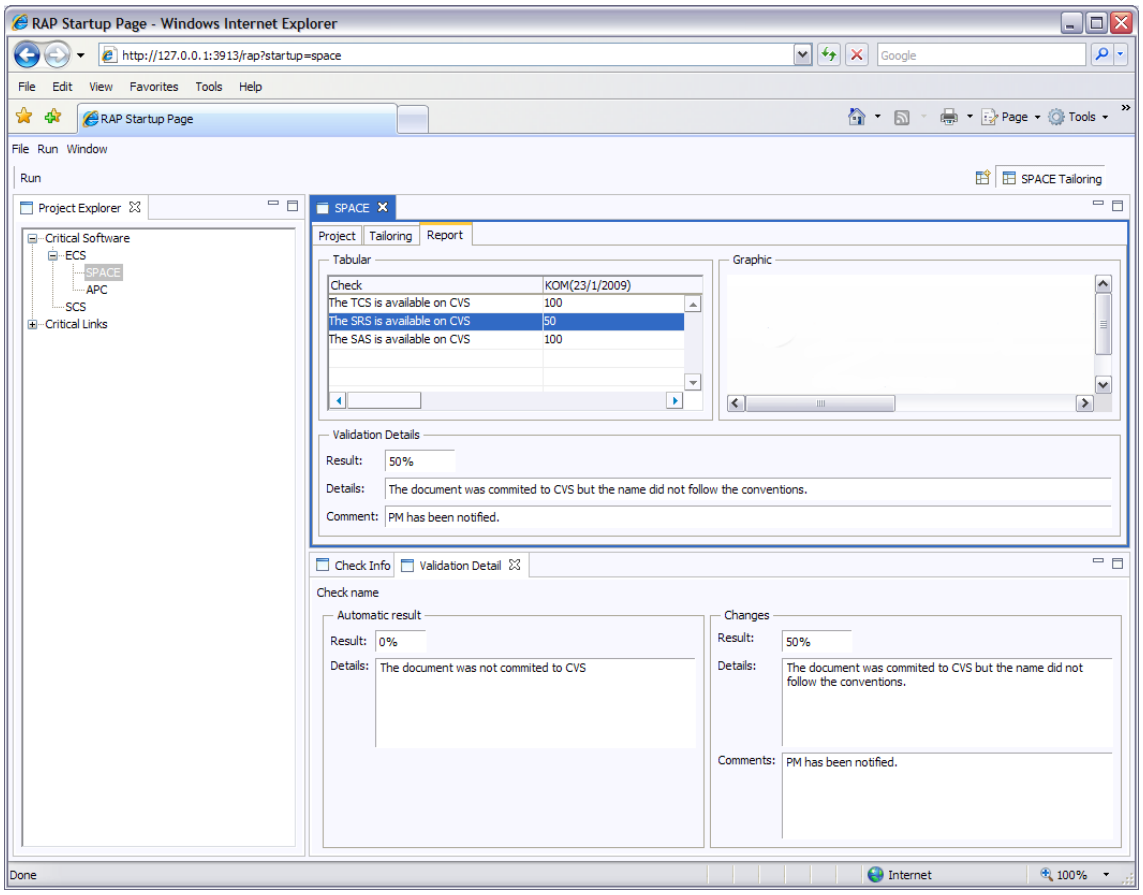

Figure 4.2: SPACE - Report

The main editor window is now used for editors of each of the domain entities, where each may be created, updated or disabled.

While the introduction of a new non-automatable verification would be straightforward, by simply creating a new Check with a null automation, the workflow for creating an automated verification would follow these steps:

1. The SPA team defines the requirements for the automation and passes them to a development team;
2. The development team implements the automation following the specified interface for SPACE;
3. The automation is deployed to the system, which may already be in production;
4. The newly published automation becomes available for the user to choose at the creation of a new check;
5. The user creates the new check, fills in the required data and selects the appropriate automation module from the published list.

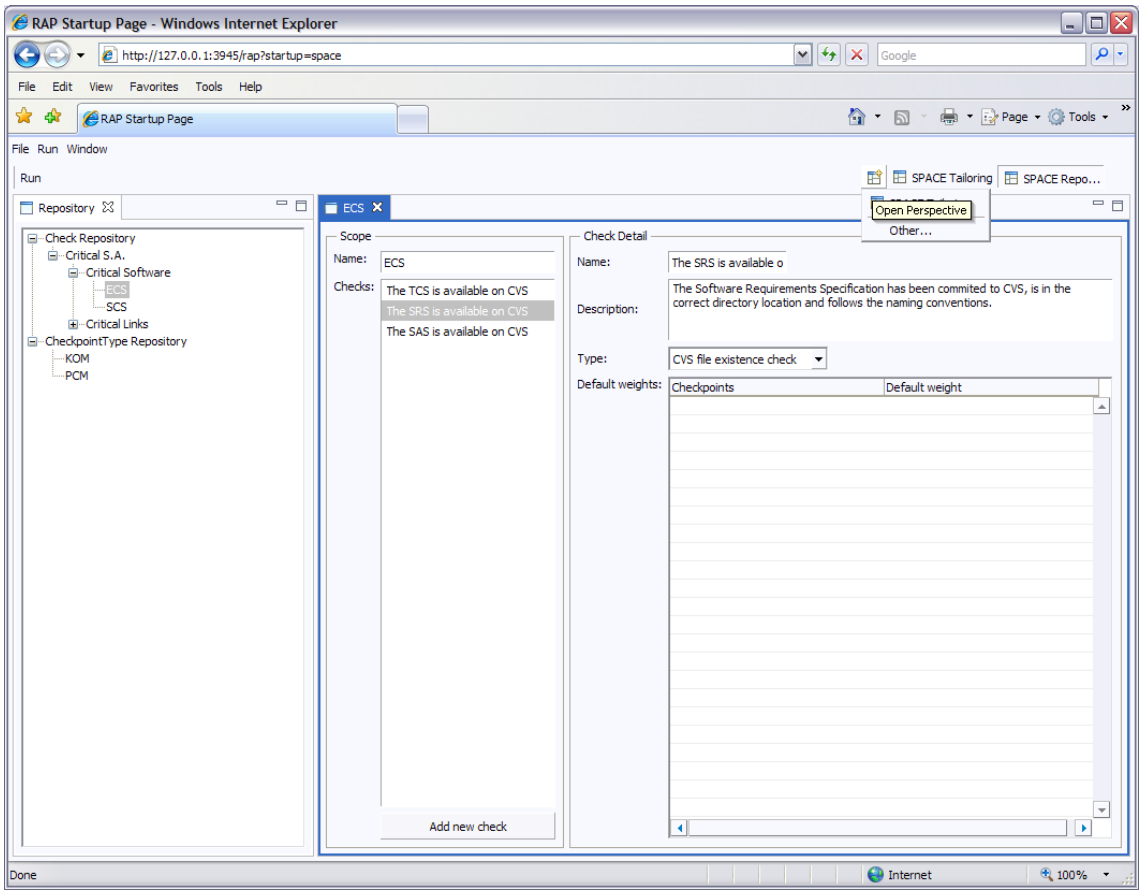

Figure 4.3: SPACE - Repository perspective

4.3 Return on Investment

The fact that SPACE is designed in a modular fashion, and is general enough to allow its application in other domains, immediately provides added value to the company by extending its tool kit with a reusable system that will optimize costs when implementing developments in the future.

However, Critical's first motivation for the development of this platform, as outlined in chapter 2, was the realisation of a considerable amount of automatable work being done by SPAEs. This problem is more easily quantifiable and allows an instant view over the potential savings SPACE may enable.

The current SPA Logbook includes 87 validations, of which 25 were evaluated as non-automatable (these are listed in appendix A. According to interviewed SPAEs, the most time-consuming validations are related to the following of conventions or lack thereof, and all the validations in this category were evaluated as automatable. Some of the non-automatable validations, however, did require careful analysis and sometimes asynchronous communication with members of the developing team, hence consuming more time.

There are no concrete metrics available concerning the relative efforts needed for each validation. However, it was considered safe to admit that the 70% of automatable validations roughly translated to 70% of the time consumed in the validation process. Considering that the usual effort allocated for a SPAE's validation tasks is 10%, we can estimate a saving of 7% of the SPAE's time. In a year, this figure translates to 140 hours per SPAE - considering the 15 SPAEs currently with Critical, this represents 2100 hours (or 52.5 weeks, more than one year) of man-work savings for the company.

This estimate does not take into account the time needed to operate SPACE. These tasks will include setting up the validations, tailoring them for each project, running them and revising the results. However, mechanisms such as default settings, automatically triggered validations and detailed generated feedback are in place to reduce the effort needed to perform these tasks. Also, the project setup and tailoring only happen in the beginning of the project. The estimated figure above therefore remains considerable.

Chapter 5

Conclusions

Taking all the above into consideration, this chapter assesses the degree of success of the project, by stepping away from the actual system and providing an overall view of the product that relates to the introductory chapter (chapter 1, Introduction). No work is done flawlessly, and as such an outline of the main issues encountered during development is also presented. Finally, it attempts to provide ideas for future improvement of the system from the privileged point of view of the implementation team.

5.1 Assessment

The development of this project was very goal oriented, especially in line with the iterative approach to the implementation. Features were prioritised and when faced with blocking issues (see section 4.1) context switching to another requirement that needed to be implemented was immediate.

As a consequence, the overall result is positive, with most of the functionality defined in the requirements having been implemented. At the time of writing, one more month was left in the project's plan, granting room for further functionality to be completed, namely the features waiting for resolution from third parties and more automated validations.

Therefore, and according to the objectives, the resulting product is a flexible and modular automation system, with proof-of-concept automations demonstrating its functionality, and is a platform for further tools to be developed upon.

As demonstrated in the section 4.3 (Return on Investment), this system potentially saves Critical 2100 hours (or 52.5 weeks, more than one year) of man-work. This is a very considerable feature, especially in a global context of economic recession as the one

faced at the date of this document, where every optimisation of costs is extremely valuable for a company to remain competitive and, ultimately, in business.

5.2 Major Issues

This result was not achieved easily, and in fact several factors hindered its development. It was an ambitious project, but one may believe more could have been done if certain obstacles had not been present.

Section 4.1 Blocking Issues referred two important external technological factors that were still unresolved at the time of writing. One other factor that slowed production was the fact that one of the main technologies, RAP, is still not in a mature state. More frequently than desirable, features that provided value to the project were not implemented or had latent limitations. Towards the end of the project, however, the releases became more and more complete towards maturity.

These technical difficulties were sometimes overcome with the help of colleagues at Critical. However, most of the know-how in this project's technologies is concentrated at the Critical's headquarters in Coimbra, which slowed down communication and therefore the solution of those problems. This also proved true in the design phase, when feedback over decisions had to be asynchronous.

These colleagues did go out of their way to help the development project, and it would even be unfair to ask for more. It is true, however, that these people are 100% allocated to their tasks and this project meant extra effort even to those directly connected to it, such as the project manager. Although it sometimes meant deviating from the plan, much team effort and group spirit was demonstrated to overcome time shortages of people with higher responsibilities, and much was learned in doing so.

5.3 Future Work

At the time of writing, the plan accounted for one more month of development work.

The roadmap included the resolution of the issues mentioned in section 4.1, the implementation of an authentication module and the automation of several validations.

Apart from the predicted work, other possible improvements arose. In terms of usability, future developments in the presentation technology (RAP) may allow for better reworks of the current interface, for example implementing drag-and-drop in certain use-cases where it could be beneficial.

One possibly valuable feature, since all the automatable validations deal with some sort of data and necessarily need to know where that data is located, would be to provide the expert with that location when viewing the detail of the validation. This would further decrease the effort needed by the expert to revise the results provided by the system.

Finally, the project plan did not include the development of a help system for the platform. Such a feature would prove valuable to the SPAE not only during training by making the learning curve more mild but also as a reference in production.

References

[Cri09] Critical. Critical software milestones, February 2009. Available at http://www.criticalsoftware.com/milestones.html.

[Ecl09a] Eclipse. Eclipse announces eclipselink project to deliver jpa 2.0 reference implementation, February 2009. Available at http://www.eclipse.org/org/press-release/20080317_Eclipselink.php.

[Ecl09b] Eclipse. Eclipse rap project, February 2009. Available at http://www.eclipse.org/rap/.

[Ecl09c] Eclipse. Eclipse rap project - release history, February 2009. Available at http://www.eclipse.org/rap/noteworthy/.

[Ecl09d] Eclipse. Eclipselink project, February 2009. Available at http://www.eclipse.org/eclipselink/.

[Ecl09e] Eclipse. Equinox project, February 2009. Available at http://www.eclipse.org/equinox/.

[EG94] et al. Erich Gamma. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, First edition, 1994.

[IBM09] IBM. Rap demo application - ibm tutorial, February 2009. Available at http://www.ibm.com/developerworks/opensource/library/os-eclipse-richajax1/?S_TACT=105AGX44&S_CMP=ART.

[JM05] Jean-Michel Lemieux Jeff McAffer. *Eclipse Rich Client Platform: Designing, Coding, and Packaging Java Applications*. Addison Wesley Professional, First edition, 2005.

[MBC06] Sandy Shrum Mary Beth Chrissis, Mike Konrad. *CMMI: Guidelines for Process Integration and Product Improvement*. Addison Wesley Professional, 2nd edition, 2006.

[Mic09] Sun Microsystems. Java persistence api, February 2009. Available at http://java.sun.com/javaee/technologies/persistence.jsp.

[OSG07a] OSGi. Osgi service platform, release 4 - core specification. Technical report, The OSGi Alliance, April 2007.

[OSG07b] OSGi. Osgi service platform, release 4 - service compendium. Technical report, The OSGi Alliance, April 2007.

[OSG09a] OSGi. Osgi alliance members, February 2009. Available at http://www.osgi.org/About/Members.

[OSG09b] OSGi. Osgi technology description, February 2009. Available at http://www.osgi.org/About/Technology.

[Qoo09] Qooxdoo. Qooxdoo project, February 2009. Available at http://qooxdoo.org/about/framework.

Appendix A

Checklists

This appendix details the lists of verifications being used in the On-Process validation at Critical Software, organised by the respective process. For each list item, an evaluation of the possibility to automate the verification, as made by the SPA team in the beginning of the project, is also presented.

A.1 Generic

- **Work Orders resulting from Audits are closed** - Check in WOW (work order management system at Critical) if all Work Orders submitted are closed;
- **Appropriate templates were used** - Not verifiable automatically.

A.2 Project Management

- **The change management activity is being managed** - Not verifiable automatically;
- **Project Opened in WISE tool** - Check if the project is opened in WISE (Critical's Information System) tool;
- **Intradoc Reference created** - Check in WISE;
- **Project Folder was created (mandatory contents included)** - Not verifiable automatically;
- **Responsibilities were identified and presented to the team** - Not verifiable automatically;
- **Master plan Baselined** - Check in WISE;
- **Case Study (Press release section filled)** - Check if Case Study is in CVS;
- **Project Infrastructure was created** - Check in WISE;
- **CVS structure created** - Check in CVS;
- **WO Group created** - Check in WOW;

- **Project Mailing List Created** - Check in WISE;
- **Project Created in JIRA** - Check in JIRA (Critical's issue tracker);
- **Enterprise Architect Database created** - Check in EA (Critical's software architecture tool);
- **Case study was reviewed** - Not verifiable automatically;
- **Case study was created** - Check in CVS;
- **Case study was approved and handed to Business Development Department** - Not verifiable automatically;
- **Finished project was closed and archived** - Check if WISE;
- **The responsible(s) for the Project Closure tasks was identified** - Not verifiable automatically;
- **Master/Detail Plan is updated** - Not verifiable automatically;
- **Meeting minutes are being produced** - Check in CVS;
- **Monthly progress reports are being produced** - Check in CVS;
- **Tasks (tickets) are being correctly managed** - Not verifiable automatically;
- **Licences/tools/facilities available and in use** - Not verifiable automatically;
- **Stakeholders are documented** - Not verifiable automatically.

A.3 Project Incident

- **The change management activity is being managed** - Not verifiable automatically.

A.4 Risk Management

- **Risks are being managed** - Check in CVS.

A.5 Subcontracting

- **Subcontrators are being controled** - Check in CVS for meeting minutes from meetings with subcontractor.

A.6 Quality Management

- **QAP approved** - Check if QAP is in CVS and status is approved;
- **Product Assurance Reports are being produced** - This will be produced by the tool;
- **QAP is updated** - Not verifiable automatically.

A.7 Configuration management

- **Configuration Item Log approved** - CIL filled with all Configuration item identified;
- **Configuration Management plan approved** - CIL filled with Configuration Management activities;
- **Code was baseline before testing** - Check in CVS;
- **Source Code was baselined** - Check in CVS;
- **CVS (conventions, local folders regularly committed, commit doesn't break the build)** - Check in CVS;
- **Deliverables were baseline before release** - Check in CVS;
- **Configuration Management plan is updated** - Check in CVS;
- **Configuration Items List is updated** - Check in CVS;
- **Configuration Audits Performed for Milestones** - Check in CVS.

A.8 Requirements Analysis

- **Requirements are approved** - Check in CVS;
- **EA File with SR status proposed** - Check in EA Database;
- **Traceability is established** - Check traceability UR-SR;
- **Prototypes were used and validated** - Not verifiable automatically;
- **UML diagrams were used to clarify requirements** - Not verifiable automatically.

A.9 Reuse

- **Reuse Analysis performed** - Check in CVS;
- **Reuse analysis meeting performed before PDR and RSU produced** - Check in CVS;
- **Reuse analysis meeting was performed before QR and RSU produced** - Check in CVS.

A.10 Software Design

- **SAS (Software Architecture Specification) approved** - Check for SAS in CVS with status approved;
- **EA File ICS status proposed** - Not verifiable automatically;
- **Interface Control Specification approved** - Check for ICS in CVS;
- **Database Specification** - Check in CVS;
- **Architectural constraints are traceable to requirements** - Check traceability.

A.11 Software Construction

- **Detailed Design is documented** - Check for DDS in CVS;
- **Code conventions are being followed** - Check in CVS.

A.12 Software Testing

- **Test Case Specification (TCS) is approved** - Check if TCS is in CVS with status approved;
- **Test Plan is approved (Plan unit,acceptance, system, integration test)** - Check if Test Plan is in CVS with status approved;
- **Environment is ready before testing phase** - Not verifiable automatically;
- **Unit Test Specification is defined** - Check for UTS in CVS;
- **Unit Tests implemented and results were documented** - Not verifiable automatically;
- **Tests (acceptance, system, integration) planned and updated** - Check in CVS;
- **Tests are traceable to all requirements** - Check traceability;
- **Tests results were recorded (all tests considered)** - Check if Test Report is in CVS;
- **Tests problems were recorded** - Check in JIRA;
- **Failed tests were corrected and verified** - Check in JIRA;
- **Testing strategy was followed** - Not verifiable automatically;
- **Test Case specification (TCS) was reviewed and updated** - Check in CVS;
- **Test Plan is updated** - Check in CVS.

A.13 Verification

- **Requirements were reviewed** - Check if Design Review Report for requirements is in CVS;
- **Source Code was reviewed** - Check for Code Review Report in CVS;
- **Architecture Specification was reviewed** - Check for Architecture Review Report in CVS;
- **Review Strategy is defined and updated** - Check in CVS;
- **Review Records are being produced** - Check Review reports in CVS;
- **Review Records are signed and placed in Project Folder** - Not verifiable automatically.

A.14 Delivery

- **Release Strategy is defined** - Check if CIL is in CVS, and Release Strategy is defined in QAP;
- **Installation Manual is approved** - Check in CVS;
- **User Manual is approved** - Check in CVS;
- **Data Package is ready for release** - Not verifiable automatically;
- **Installation and User Manual was reviewed** - Check in CVS;
- **Statement of Acceptance received** Check in CVS;
- **Installation report was approved** Check in CVS;
- **Release sign-off form was correctly filled, stored in the Project Folder and signed by the PM and SQA** - Not verifiable automatically;
- **Release Strategy is being followed** - Not verifiable automatically;
- **Release environment is defined in Release sign-off form** - Not verifiable automatically;
- **Release notes with major changes are being delivered to the client** - Not verifiable automatically.

A.15 Lessons Learned

- **Lessons Learned were recorded** - Check in WISE.