UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

GUIDELINE TO APPLY THE ISO 90003:2004 STANDARD TO SMEs OF SOFTWARE DEVELOPMENT

INGENIERÍA TÉCNICA INFORMÁTICA DE GESTIÓN
PROYECTO FINAL DE CARRERA

Autor: Ginés Sánchez Bustillo
Tutor: Antonio García Carmona
Noviembre, 2010

UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

GUIDELINE TO APPLY THE ISO 90003:2004 STANDARD TO SMEs OF SOFTWARE DEVELOPMENT

INGENIERÍA TÉCNICA INFORMÁTICA DE GESTIÓN
PROYECTO FINAL DE CARRERA

Autor: Ginés Sánchez Bustillo
Tutor: Antonio García Carmona
Noviembre, 2010

EL TRIBUNAL

Presindente:_____

Vocal:_____

Secretario:_____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día **17** de **Noviembre** de **2010** en Colmenarejo, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de:

VOCAL

SECRETARIO **PRESIDENTE**

INTRODUCTION

Alice, who is holding hands by the Queen to run together, are quite surprised to see that, although run as fast as they can, its surroundings do not seem to change.
...

Alice looked round her in great surprise. `Why, I do believe we've been under this tree the whole time! Everything's just as it was!'

`Of course it is,' said the Queen, `what would you have it?'

`Well, in our country,' said Alice, still panting a little, `you'd generally get to somewhere else -- if you ran very fast for a long time, as we've been doing.'

`A slow sort of country!' said the Queen. `Now, here, you see, it takes all the running you can do, to keep in the same place. If you want to get somewhere else, you must run at least twice as fast as that!'

Lewis Carrol, Through the looking glass and what Alice found there.

The quality, based on continuous improvement, is the surest way to encourage business continuity. In an evolving system, quality and continuous improvement are essential to maintaining the condition which has been reached, not as a form of improvement.

ACKNOWLEDGMENTS

- To my family.

- To my company, Almira Labs., because the knowledge is all that matters and I have a lot of fun working with you.

- To my Project Advisor, Antonio García Carmona.

Thanks to all!

INDEX

INDEX OF PICTURES

CHAPTER I

I - SCOPE

1. Foreword

Nowadays, markets around the world are increasingly characterized by high dynamism, globalization and competition. The competitiveness of the products is a function of the capacity to satisfy customer's requirements, the main goal of any organization, who base its decisions on two factors: price and, especially, quality. As a result of changes in the social and economic context, the acquisition and recognition of adequate quality to the possibilities of the organization has become a priority for senior management.

Get a definition of quality is not complicated, especially when it sits in a goal as simple as is the full satisfaction of the needs and expectations of customers in a very efficient and profitable as possible; it can be really a complex implantation, and also there is no standard model which applies to any organization.

Despite the advantages of gamble for total quality, often we find organizations, especially small and medium enterprises (hereinafter *SMEs*), whose first contact with the quality is through the systems defined by *ISO 9000* standards. While is true that the overall quality is much more than meet the requirements of a particular standard, a quality management system standard is one of the components that can help to produce a successful environment of total quality.

The exponential growth experienced by the number of certified companies in *ISO 9000* standards shows that, despite the high cost for proper introduction and recognition of a quality system, the benefits, mainly in the medium and long term, outweigh the initial economic effort.

However, it is also true that the *ISO 9000* standards are only the beginning, provide mechanisms with which to carry out a systematic improvement, but failed to improve the operation by themselves. People are taking things that motivate ambition, not doing the same practices mundane, day after day.

2. Goals

By undertaking this Final Career Project I have tried to accomplish two main goals:

- The main goal has been adapt the *ISO 90003:2004* standard to SMEs software development company (purposing norms, procedures,

work instructions for every one of the guideline sections), and as an example I have created a Web application named *Booking Books*.

• The second goal has been, by developing the software product with a level of quality accepted and recognized, get certified in the *ISO 9001:2008* standard. I hope that all software development SMEs take a firm decision to adapt the standard *ISO 9001:2008* or other standard of quality, in order to motivate them to improve their quality and thus be more competitive. The certification is not only a necessity in terms of quality; the standard can keep out of the business to the weaker companies, as it is becoming a prerequisite to have certifications in order to do business.

3. Problems Faced By SME

I am aware of the limited resources available to an SME or another newly created company in key process areas as could be:

• **Organization and development:** lack of formal structure, lack of systematization of its development processes operations and activities, lack of written policies, lack of supervision and performance standards, lack of measurement and quality control, poor distribution of work; I have described for every step of the software life cycle a guideline to correctly perform the phases of the software development process.

• **Human resources:** lack of training, excessive staff turnover; I have described a human resources training and recruitment template and all the staff responsibilities.

• **Marketing:** failure to use merchandising techniques to publicize the product and bring it to the needs of the consumer; this section is out of the scope of this Project.

• **Credit problems:** lack of access to financial credit, excessive cost of proprietary tools; I have purposed open source programs (both in basic software such an operating system, and tools which support the software development life cycle), in order to reduce the *Total Cost Ownership* (TCO), reducing the impact of proprietary tools in the company accounting.

• **Lack of innovation and improvement of processes;** I have purposed a process maturity model in order to improve the maturity of the processes.

4. Project Structure

This project has four parts:

In the first part, which includes chapters **I - SCOPE** and **II - KEY CONCEPTS**, I explain the scope and key concepts of this project, with a theoretical explanation of what the quality is. In order to understand this project, first is necessary read them.

In the second part, which include chapter **III - GUIDELINE TO APPLY THE ISO 90003:2004 TO A SMEs OF SOFTWARE DEVELOPMENT**, I have uses the *ISO 9001:2008* and its guideline for the software development *ISO 90003:2004*, to implement their definitions to a SME of software development, so that the company will be able to get certified in the *ISO 9001:2008* standard (*ISO 90003:2004* is not a certifiable standard).

In the third part, which include chapter **IV - BOOKING BOOKS APPLICATION**, I have created an example of software development, a Web Application (an online library named *Booking Books*), developed in J2EE, so reader will see in a practical way, how they should perform all the tasks of the software development process, from the first meeting with the client, to the final delivery of the software product.

Finally, I have added the chapter **V – CONCLUSIONS** and **ANNEX I: SOURCE CODE** with the source code of the most important classes and **ANNEX II: PROJECT EXECUTION** with the time needed to develop this project and the meeting minutes.

5. Web Applications Overview

In software engineering, a **Web Application** (hereinafter **WebApp)** is an application that is accessed via web browser over a network such as the Internet or an intranet. It is also a computer software application that is coded in a browser-supported language (such as *HTML*, *JavaScript*, *Java*) and reliant on a common web browser to render the application executable.

Web applications are popular due to the ubiquity of a client, sometimes called a thin client. The ability to update and maintain web applications without distributing and installing software on potentially thousands of client computers is a key reason for their popularity. Common web applications include web mail, online retail sales, online auctions, massively multiplayer online role-playing games and many other functions.

I have used the ***JAVA* Programming Language** (*J2EE*), because is the most commonly used in this object oriented applications environment. I have tried to make the project without dependencies with a concrete language code, so that reader will be able to use this guide lines with other similar object oriented Web technologies like *Microsoft .NET*, or *PHP Hypertext Processor*.

6. Document Definition Template

The next document definition template specifies every the format of this project:

ISO 9001:2008 Text

In this section is the *ISO 9001:2008* Standard.

The *ISO 9001:2008* describes the requirements of a quality management system. This standard can be applicable to all type of companies.

NOTE: The *ISO 90003:2004* standard is based on the *ISO 9001:2000*; however, this project has been done using the *ISO 9001:2008* because is the latest *ISO 9001* standard.

ISO 90003:2004 Guideline

In this section is the *ISO 90003:2004* Text.

The ISO *90003:2004* describes a guideline for organizations in the application of *ISO 9001:2008* to the acquisition, supply, development, operation and maintenance of computer software. This guideline is applicable to any software development company, regardless of its type of software provided, size or type.

Guideline Application

1. Description

In this section, I will write the guidelines of both standards *ISO 9001:2008* and *ISO 90003:2004,* applying them to a SME of software development.

If the company follows this guide, it should be able to get certified as *ISO 9001:2008* compliant.

Guideline Application Tool

1. Description

In this part of the document (when applicable), I will select some tools and frameworks for the guideline application.

CHAPTER II

II - KEY CONCEPTS

1. Definition of Quality

The industry has proposed different definitions of the word **quality**, some of them are:

1. **ISO 9000:** degree to which a set of inherent characteristic fulfills requirements. The standard defines requirement as need or expectation.
2. **Six Sigma:** number of defects per million of opportunities. The metric is tied in with a methodology and a management system.
3. **Philip B. Crosby:** conformance to requirements. The difficulty with this approach is that the requirements may not fully represent customer expectations; Crosby treats this as a separate problem.
4. **Joseph M. Juran:** fitness for use. Fitness is defined by the customer.
5. **Noriaki Kano:** presents a two-dimensional model of quality: must-be quality, and attractive quality. The former is near to the fitness for use and the latter is what the customer would love, but has not yet thought about. Supporters characterize this model more succinctly as: products and services that meet or exceed customers' expectations.
6. **American Society for Quality:** a subjective term for which each person has his or her own definition. In technical usage, quality can have two meanings: the characteristics of a product or service that bear on its ability to satisfy stated or implied needs; or a product or service free of deficiencies.
7. **Peter Drucker:** quality in a product or service is not what the supplier puts in. It is what the customer gets out and is willing to pay for.
8. **Roger S. Pressman:** degree to which a system, component or process meets the specified requirements and the needs and expectations of the client or user.

In providing products or services, there are three basic parameters that determine its sale: price, quality and distribution. Customers want their products and services are distributed on a given quality, within a specified time and at a price commensurate with its actual value. An organization will survive only if it creates and keeps customers satisfied, and this will only succeed if you sell products or services that meet customer expectations.

The quality is determined by the ability of a product or service to satisfactorily serve the purposes of the user during use.

The final arbiter in matters of quality is the customer. He is the only one who can decide whether the quality of products and services to be delivered are satisfactory. Ultimately, the survival of an organization depends solely on its ability to offer products that meet the needs of their customers and even beyond, which will make the organization makes a profit for survival.

2. ¿What is a Quality Management System?

A quality management system (QMS) can be expressed as the organizational structure, procedures, processes and resources needed to implement quality management, in the pursuit of customer satisfaction.

The main elements are:

Organization Structure. The organizational system of the company with a hierarchy of management and executive levels.

The structure of responsibilities. The structure of responsibilities involves people and departments.

Procedures. The procedures meet the guidelines detailed permanent plan to control the actions of the organization.

Processes. Processes respond to the full sequence of operations aimed at achieving a specific goal.

Resources. The resources, not only financial, but human, technical and otherwise, must be defined on a stable and also of being so circumstantial.

These five sections are not always clearly defined and in a company. *Walter A. Shewhart* defined that good management is to make a mistake once in a while and then another, but also from time to time. He realized what he had to do is put in place rules by which errors are reduced and therefore the economic loss, making a system of quality management.

Any quality management system of an organization has as a base the quality manual and is supplemented by a number of additional documents such as manuals, standards, procedures, methods of operating, technical instructions, records and information systems.

The company or organization which complies with quality proposal, must be involved in its entirety, but in turn requires an attitude of continuous improvement based on four steps (*PDCA*), which are *Planning* (identify processes, gather data, set goals), *Perform* (executing the processes defined in the previous step), *Check* (after a while, collect data and compare it with the initial objectives) and *Act* (redesign processes improvements)... and start again with the whole process (continuous improvement).

3. ISO Quality Standards

ISO 9000 are a series of international standards for quality systems. Specifies the recommendations and requirements for the design and evaluation of a quality management system. *ISO 9000* is not a standard product. It does not contain any requirement by which a product or service have to meet, there is no criterion for acceptance of the product. The requirements and recommendations apply to organizations that provide products or services, and therefore affect the way that goods and services are designed, manufactured or installed.

The family of *ISO 9000* standards have been developed to assist organizations, regardless of their size or business sector in the implementation and operation of systems for effective quality management. The *ISO 9000* standards related with the purpose of this project are:

- **ISO 9000**, which describes the fundamentals of the quality management systems and specifies the terminology.
- **ISO 9001**, which specifies requirements for the quality management system for organizations that need to demonstrate their ability to provide products that meet the requirements of its customers and regulations that apply, and its aim is to increase customer satisfaction customer.
- **ISO 90003**, which is a guideline for the applications of ISO 9001 to software computer. This project is centered in it.

3.1. Why Get an ISO Certification?

An *ISO* certification is designed to determine whether a company has the capacity to meet the requirements of its customers. The results are obtained through sampling of activities, documents and products or services. The granting of a certificate does not mean you do not have any non-compliance; means that it has not found any significant.

Some of the advantages of obtaining a certificate are as follows:

1. The company will be considered a company with a high level of quality, and therefore any customer seeking a specific service can discover and contact the company.
2. Allows you to announce that the company is registered in *ISO 9001*, and this helps your profile exposure and marketing.
3. The company will be able to bid in competition restricted to organizations *ISO 9001* compliant.

Once the company gets the *ISO 9001* certification, may prove to their customers that is serious regarding quality. Obtaining the certificate is easier than maintaining it.

3.2. Main Requirements

There are some requirements that most organizations fail to meet, and they should lead the essential effort. Other requirements are important and will fail if there are deficiencies in them.

The following seven elements usually require new methods:

1. **Responsibility for direction**, particularly with regard to the commitment and management review.
2. **System of quality**, particularly with regard to documentation and maintenance.
3. **Control of documents**, particularly with respect to change control and obsolescence.
4. **Teams of inspection**, measuring and testing, particularly with regard to what is calibrated, calibration frequency and the action required following the obtaining of a result out of calibration.
5. **Corrective actions**, particularly with respect to analyzing risk, prevent recurrence of nonconformities, change procedures.
6. **Register of quality**, particularly with respect to records that are established and maintained, and how long they are retained.
7. **Internal audits**, particularly with regard to planning, implementing and monitoring plan for corrective action.

3.3. Reviews to ISO 9000 Standards

ISO 9000 standards have become the basis for trust between suppliers and customers to avoid the proliferation of rules and instructions that each organization imposes on its suppliers. However, not all opinions about *ISO 9000* standards are favorable, perhaps the most frequent criticism relates to generating excessive bureaucracy and the possibility of purely formal compliance with same, some others are:

• are assumed by the industry, which have been given a somewhat particular bias;

• are based on the activities of the organization to have impact on the quality of their products, leaving out essential aspects to achieve effective and efficient management of the organization;

• its target is reduced to ensure a uniform and stable level of quality of products and services. It is limited to ensure the achievement of certain pre-established minimum quality

• systematization proposed by *ISO 9000* may lead to bureaucratic processes that limit the creativity and initiative of the organization, and can also mean the loss of flexibility characteristic of SME.

The *ISO 9000* model looks only inside the company, at best considers the integrated supplier within the supply chain, and customer

as the recipient of the shares, but considering the results of the company in customer satisfaction in projection to the society or the prospects of business's viability.

3.5. ISO 9001 Standards Specification

3.4.1. ISO 9000:2005 Standard

ISO 9000:2005 describes fundamentals of quality management systems, which form the subject of the ISO 9000 family, and defines related terms.

3.4.2. ISO 9001:2008 Standard

The *ISO 9001:2008* specifies the requirements for a quality management system that can be used for its implementation by the organizations, for certification or contractual purposes. It focuses on the effectiveness of quality management system, to comply with customer requirements.

This standard specifies requirements for a quality management system where an organization:

- Need to demonstrate its ability to provide products that meet customer requirements and applicable regulations
- Aims to enhance customer satisfaction through the effective implementation of the system.

1. **Quality management system.**

Included in this section are the general requirements and documentation requirements.

For the section on general requirements, the organization shall establish, document, implement and maintain a quality management system and continually improve its effectiveness: identifying the necessary processes, determining their sequence and interaction, determining criteria to ensure they are effective, ensuring availability resources and information, by monitoring, measuring and analyzing these processes, and implement actions needed to achieve the results and continuous improvement of processes.

For documentation requirements, a quality management system must include a quality manual, document control and records control.

- **Management Responsibility.**

Included in this section is the management commitment (must provide evidence of the same with the development and implementation of the system of quality management and continuous improvement in efficiency), customer focus (you must ensure that Client requirements identified and met), quality policy (which is properly understood, revised), planning (the objectives must be measurable and enforceable)

responsibility, authority and communication (the responsibilities and authorities must be defined and reported in the organization), and management review (to ensure its continuing suitability, adequacy and effectiveness).

- **Resource Management.**

Included in this section are the provision of resources (to implement, maintain and improve the system of quality management), human resources (education, training, skills and relevant experience), infrastructure (buildings, workspace and associated services, equipment for processes and support services) and work environment (creating a good work environment for compliance with the requirements of the product).

- **Product realization.**

Included in this planning product realization (with monitoring, verification, validation, inspection and testing / test), Customer-related processes (with the requirements specified by the client, unspecified but necessary legal and regulatory as well as any additional one), design and development (with the reviews, verifications and validations in each), shopping (selecting suppliers based on their ability to supply products), production and service provision (with their availability and use of tracking devices measurement), and control of devices for monitoring and measuring (to provide evidence of product conformity with certain requirements).

- **Measurement, analysis and improvement.**

Included in this generality, monitoring and measuring (through audits planned), control of nonconforming product (to prevent unintended use or delivery), data analysis (to demonstrate the suitability of the system and assess where you can make possible improvement) and improvement (including information gathered previously to eliminate causes of nonconformities).

3.4.3. ISO 90003:2004 Standard

ISO 90003:2004 provides guidance for organizations in the application of *ISO 9001:2008* to the acquisition, supply, development, operation and maintenance of computer software.

It identifies the issues which should be addressed and is independent of the technology, life cycle models, development processes, sequence of activities and organizational structure used by an organization. The guidance and identified issues are intended to be comprehensive but not exhaustive. Where the scope of an organization's activities includes areas other than computer software development, the relationship between the computer software elements of that organization's quality management system and the remaining aspects should be clearly documented within the quality management system as a whole.

CHAPTER III

III - GUIDELINE TO APPLY THE ISO 90003:2004 TO A SME OF SOFTWARE DEVELOPMENT

1. Scope

1.1 General

ISO 9001:2008 Standard

This International Standard specifies requirements for a quality management system where an organization

a) needs to demonstrate its ability to consistently provide product that meets customer and applicable regulatory requirements, and

b) aims to enhance customer satisfaction through the effective application of the system, including processes for continual improvement of the system and the assurance of conformity to customer and applicable regulatory requirements.

NOTE 1 In this International Standard, the term "product" only applies to

a) product intended for, or required by, a customer,

b) any intended output resulting from the product realization processes.

NOTE 2 Statutory and regulatory requirements can be expressed as legal requirements

ISO 90003:2004 Guideline

This International Standard provides guidance for organizations in the application of *ISO 9001:2008* to the acquisition, supply, development, operation and maintenance of computer software and related support services. It does not add to or otherwise change the requirements of *ISO 9001:2008*.

Annex A (informative) provides a table pointing to additional guidance in the implementation of ISO 9001:2000 available in ISO/IEC JTC 1/SC 7 and ISO/TC 176 standards.

The guidelines provided in this International Standard are not intended to be used as assessment criteria in quality management system registration/certification.

Guideline Application

1. Description

It is not necessary add any guideline. Only advice that a good level of quality is an issue which involves all the organization; it could require years to achieve them.

Organizations must be aware about the necessity and importance of develop software with a high level of quality and the importance of the quality at medium and long term in relation to the customer's satisfaction.

Guideline Application Tool

1. Description

It is not necessary add any guideline application tool.

1.2 Application

ISO 9001:2008 Standard

All requirements of this International Standard are generic and are intended to be applicable to all organizations, regardless of type, size and product provided.

Where any requirement(s) of this International Standard cannot be applied due to the nature of an organization and its product, this can be considered for exclusion.

Where exclusions are made, claims of conformity to this International Standard are not acceptable unless these exclusions are limited to requirements within clause 7, and such exclusions do not affect the organization's ability, or responsibility, to provide product that meets customer and applicable regulatory requirements.

ISO 90003:2004 Guideline

The application of this International Standard is appropriate to software that is

- part of a commercial contract with another organization,
- a product available for a market sector,
- used to support the processes of an organization,
- embedded in a hardware product, or
- related to software services.

Some organizations may be involved in all of the above activities; others may specialize in one area. Whatever the situation, the organization's quality management system should cover all aspects (software related and non software related) of the business.

Guideline Application

1. Description

The *ISO* Standards are generic, so the organizations must adapt these standards to their products. In this project I will apply this standard to a SME software development company. I have developed, too, a WebApp as an example using the *J2EE* technology.

I really have done an effort to give more importance to software engineering concepts (software life cycle, configuration management, design patters) than concepts related to a concrete implementation technology (*J2EE* technology, development frameworks as *Struts* or concrete test tools). Considering it is possible to develop this project in other similar object-oriented programming language, with other development frameworks, other tools, and other technologies (*.Net* or *PHP*, for example).

A software project can be a success or a total failure regardless the features of a technology; but with a set of mature rules, standards, procedures, and software engineering process, is more probably to achieve the project success.

So, despite I have created a Web Application as an example (so you will be able to see and have the know-how in a practical way), please, always pay more attention and be centred in the concepts related to the software engineering.

Guideline Application Tool

1. Description

It is not necessary add any guideline application tool.

2. Normative references

ISO 9001:2008 Standard

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 9000:2005, Quality management systems – Fundamentals and vocabulary

Guideline Application

1. Description

In *Chapter V – Conclusions*, I have added the section *3. Referenced Resources* with the some referenced documents.

Guideline Application Tool

1. Description

It is not necessary add any guideline application tool.

3. Terms and definitions

ISO 9001:2008 Standard

For the purposes of this International Standard, the terms and definitions given in *ISO 9000* apply.

Throughout the text of this International Standard, wherever the term product occurs, it can also mean service.

ISO 90003:2004 Guideline

For the purposes of this document, the terms and definitions given in ISO 9001:2008, and certain terms (repeated here for convenience) given in ISO 12207 apply.

However, in the event of a conflict in terms and definitions, the terms and definitions specified in ISO 9000:2005 apply.

3.1: **Activity:** collection of related tasks.

3.2: **Baseline:** formally approved version of a configuration item, regardless of media, formally designated and fixed at a specific time during the configuration item's life cycle.

3.3: **Configuration item:** entity within a configuration that satisfies an end use function and that can be uniquely identified at a given reference point.

3.4: **COTS (Commercial-Off-The-Shelf):** software product available for purchase and use without the need to conduct development activities

3.5: **Development:** software life cycle process that contains the activities of requirements analysis, design, coding, integration, testing, installation and support for acceptance of software products.

3.6: **Life cycle model:** framework containing the processes, activities, and tasks involved in the development, operation, and maintenance of a software product, spanning the life of the system from the definition of its requirements to the termination of its use

3.7: **Measure, verb:** make a measurement.

3.8: **Measure, noun:** variable to which a value is assigned as the result of measurement.

3.9: **Measurement:** set of operations having the object of determining a value of a measure.

3.10: **Process:** set of interrelated or interacting activities which transform inputs into outputs.

3.11: **Regression testing:** testing is required to determine that a change to a system component has not adversely affected functionality, reliability or performance and has not introduced additional defects

3.12: **Release:** particular version of a configuration item that is made available for a specific purpose

3.13: **Replication:** copying a software product from one medium to another

3.14: **Software item:** identifiable part of a software product

3.15: **Software product:** set of computer programs, procedures, and possibly associated documentation and data.

3.16: **Software service:** performance of activities, work, or duties connected with a software product, such as its development, maintenance and operation.

Guideline Application

1. Description

Some terms and definitions to be used extensively in this project are as follows:

- **Abstraction:** the essential characteristics of an entity that distinguish it from all other kinds of entities. An abstraction defines a boundary relative to the perspective of the viewer. (Booch, Rumbaugh, Jacobson, 2005).

- **Actor:** the person or automated system that interacts with a product use case. Actors are also known as users or final users.

- **Architecture:** set of significant decisions about the organization of a software system, the selection of the structural elements and their interfaces by which the system is composed, together with their behaviour as specified in the collaborations among those elements, the composition of these structural and behavioural elements into progressively larger subsystems, and the architectural style that guides this organization these elements and their interfaces, their collaborations, and their composition. (Booch, Rumbaugh, Jacobson, 2005).

- **Architectural patters:** a pattern that defines a certain structure or behaviour, usually for the architectural view of a specified model. Examples are the Layer, Client-server, Three-Tier and Peer-to-Peer patterns, each of which defines a certain structure to the deployment model and also suggest how

components should be allocated to its nodes. (USDP, Booch, Rumbaugh, Jacobson, 2005).

• **Artifact:** general term for any kind of information created, produced, changed, or used by workers in developing the system. Some artifacts are UML diagrams and their associated text, user-interfaces sketches and prototypes. (USDP, Booch, Rumbaugh, Jacobson, 2005).

• **Assertion:** a formal condition describing the semantic properties of software elements, especially routines and loops. Used in expressing contracts. Assertions include in particular pre conditions, post conditions, class invariants and loop invariants. (Bertrand Meyer).

• **Audit:** checking that a configuration item released for usage meets demands—that it fulfils specifications and, when released, is complete according to configuration management information. In this book, this is regarded as a quality assurance activity. (Anne Mette Jonassen Hass, 2002).

• **Cohesion:** a measure of how strongly-related and focused the various responsibilities of a software module are. Cohesion is an ordinal type of measurement and is usually expressed as "high cohesion" or "low cohesion" when being discussed. Modules with high cohesion tend to be preferable because high cohesion is associated with several desirable traits of software including robustness, reliability, reusability, and understandability whereas low cohesion is associated with undesirable traits such as being difficult to maintain, difficult to test, difficult to reuse, and even difficult to understand.

• **Class diagram:** diagram that shows a set of classes, interfaces, and collaborations and their relationships; class diagrams address the static design view of a system; a diagram that shows a collection of declarative (static) elements. (Booch, Rumbaugh, Jacobson, 2005).

• **Client:** someone who has been impacted by the product, and therefore the same user, organization or person who receives a product.

• **Constraints:** Constraints are global requirements. They can be constraints on the project itself or restrictions on the eventual design of the product. Constraints are global issues that shape the requirements (for example, the system shall operate in whatever Internet explorer).

• **Contingency plan:** a plan describing how to act if certain risks materialize. (USDP, Booch, Rumbaugh, Jacobson, 2005).

• **Coupling:** the degree to which software components depend on each other. (Addison-Wesley 1998).

- **Design:** the act of crafting a technical solution to fit the requirements, within the constraints.

- **Design pattern:** a design pattern systematically names, motivates, and explains a general design that addresses a recurring design problem in object-oriented systems. It describes the problem, the solution, when to apply the solution, and its consequences. It also gives implementation hints and examples. The solution is a general arrangement of objects and classes that solve the problem. The solution is customized and implemented to solve the problem in a particular context. (Addison-Wesley 1998).

- **Developer:** someone who contributes to the technical development of the product. Examples can be: tester, designer or programmer.

- **Development:** process of the life cycle that includes activities of the requirements analysis, design, coding, integration, testing, installation and support for the acceptance of software products.

- **Entity:** a name in the software text that denotes a run-time value (object or reference). (Bertrand Meyer).

- **Exception:** the inability of a routine to achieve its contract through one of its possible strategies. May result in particular from a failure of a routine called by the original routine. (Bertrand Meyer).

- **Framework:** an architectural pattern that provides an extensible template for applications within a domain. (Booch, Rumbaugh, Jacobson, 2005).

- **Inheritance:** a mechanism whereby a class is defined in reference to others, adding all their features to its own. (Bertrand Meyer).

- **Instance:** an object built according to the mold defined by the class or any one of its proper descendants. See also direct instance, proper descendant and generator. (Bertrand Meyer).

- **Interaction diagram:** the view of a system's architecture that encompasses the objects, threads, and processes that form the system's concurrency and synchronization mechanisms, the set of activities, and the flow of messages, control, and data among them. (Booch, Rumbaugh, Jacobson, 2005).

- **Interface:** the set of all signatures defined by an object's operations. The interface describes the set of requests to which an object can respond. (Addison-Wesley 1998).

- **Iteration:** a distinct set of activities with a baseline plan and evaluation criteria that result in a release, either internal or external. (Booch, Rumbaugh, Jacobson, 2005).

- **Pattern:** a common solution to a common problem in a given context. (Booch, Rumbaugh, Jacobson, 2005).

- **Persistence:** The ability of a software development environment or language to make objects persistent and support the retrieval of persistent objects for use by systems. (Bertrand Meyer).

- **Policies of quality:** overall intentions and direction of an organization related to quality as formally expressed by top management.

- **Post condition:** an assertion attached to a routine, which must be guaranteed by the routine's body on return from any call to the routine if the precondition was satisfied on entry. Part of the contract governing the routine. (Bertrand Meyer).

- **Precondition:** an assertion attached to a routine, which must be guaranteed by every client prior to any call to the routine. Part of the contract governing the routine. (Bertrand Meyer).

- **Procedure:** how to carry out an activity or process.

- **Process:** set of interrelated activities or interacting, which transformed elements of entry into results.

- **Product:** result of a process.

- **Product satisfaction:** product features that correspond to the needs of a customer, which is crucial to their needs satisfaction selling.

- **Quality:** degree to which a set of inherent characteristics complies with the requirements.

- **Quality assurance:** part of quality management aimed at providing confidence that fulfil the requirements of quality.

- **Quality management:** coordinated activities to direct and control an organization with regard to quality.

- **Quality management system:** system to direct and control an organization with regard to quality.

- **Quality manual:** a document that specifies the quality management system of an organization.

- **Regression test:** the retesting of a build that was previously tested in previous builds. Regression tests are primarily done to verify that *old functionality* in *old builds* still works when *new functionality* is added in a *new build*. (USDP, Booch, Rumbaugh, Jacobson, 2005).

- **Release:** a relatively complete and consistent set of artifacts delivered to an internal or external user; the delivery of such a set. (Booch, Rumbaugh, Jacobson, 2005).

- **Requirement:** need or expectation established, generally implied or obligatory.

- **Requirement (Functional Requirement):** something that the product must do. Functional requirements part of the fundamental processes of the product.

- **Requirement (Non Functional Requirement):** a property of quality that the product must have, such as en appearance, speed, security or accuracy property.

- **Scenario:** sequence of action and interactions that occurs under certain conditions, expressed without ifs or branching. (Alistair Cockburn, 2000).

- **Software configuration management:** process that comprises factors such as configuration, identification, status accounting, review, build management, process management, that define how an organization builds and releases products, and identifies and tracks changes. (Dart 1992).

- **Software life cycle:** framework that contains the processes, activities and tasks involved in the development, operation and maintenance of a software product.

- **Stakeholder:** person or group that has an interest in the performance or success of an organization.

- **Test:** a core workflow whose essential purpose is to verify the result from implementation by testing each build, including both internal and intermediates builds, as well as final versions of the system to be released to external parties. (USDP, Booch, Rumbaugh, Jacobson, 2005).

- **Test case:** a specification of one case to test the system, including what to test with which input, result, and under which conditions. (USDP, Booch, Rumbaugh, Jacobson, 2005).

- **Test plan:** a plan that describes the testing strategies, resources and schedule. (USDP, Booch, Rumbaugh, Jacobson, 2005).

- **Traceability:** the ability to track the history, application or location of anything that is under consideration.

- **Unit acceptance test (UAT):** is black-box testing performed on a system by the customer (the user or client) prior to accepting transfer of ownership. This is also known as end-user testing, site (acceptance) testing, or field (acceptance) testing. (Managing Web Projects, Edward B. Farkas).

- **Use case:** a use case expresses a contract between the stakeholders of a system about its behaviour. It describes the system's behaviour and interactions under various conditions as it responds to a request on behalf of the stakeholders, the primary actor, showing how the primary actor's goal gets delivered or fails.

The use case collects together the scenarios related to the primary actor's goal. (Alistair Cockburn, 2000).

- **Use case driven:** in the context of the software life cycle, meaning that use cases are used as a primary artifact for establishing the desired behaviour of the system and for communicating this behaviour among the stakeholders of the system. Also meaning that uses cases are the primary input to the analysis, design, implementation, and test of the system, including the creation, verification and validation of the system's architecture. (USDP, Booch, Rumbaugh, Jacobson, 2005).

- **Validation:** confirmation by providing evidences that the requirements for a specific use or application planned.

- **Workspace:** place where a developer keeps all of the artifacts that they need to accomplish a task; can be a directory tree on disk in the developer's working area, or it can be a collection of files maintained in an abstract space by a tool. (Berczuk, 2002).

Guideline Application Tool

1. Description

It is not necessary add any guideline application tool.

4. Quality Management System

4. 1. General requirements

ISO 9001:2008 Standard

The organization shall establish, document, implement and maintain a quality management system and continually improve its effectiveness in accordance with the requirements of this International Standard.

The organization shall

a) determine the processes needed for the quality management system and their application throughout the organization (see 1.2),

b) determine the sequence and interaction of these processes,

c) determine criteria and methods needed to ensure that both the operation and control of these processes are effective,

d) ensure the availability of resources and information necessary to support the operation and monitoring of these processes,

e) monitor, measure and analyse these processes, and

f) implement actions necessary to achieve planned results and continual improvement of these processes.

These processes shall be managed by the organization in accordance with the requirements of this International Standard.

Where an organization chooses to outsource any process that affects product conformity to requirements, the organization shall ensure control over such processes. The type and extent of control to be applied to these outsourced processes shall be defined within the quality management system.

NOTE 1 Processes needed for the quality management system referred to above include processes for management activities, provision of resources, product realization and measurement, analysis and improvement.

NOTE 2 An "outsourced process" is a process that the organization needs for its quality management system and which the organization chooses to have performed by an external party.

NOTE 3 Ensuring control over outsourced processes does not absolve the organization of the responsibility of conformity to all customer, statutory and regulatory requirements. The type and extent of control to be applied to the outsourced process can be influenced by factors such as

a) the potential impact of the outsourced process on the organization's capability to provide product that conforms to requirements,

b) the degree to which the control for the process is shared,

c) the capability of the achieving the necessary control through the application of 7.4.

ISO 90003:2004 Guideline

Guidance is provided for items a) and b) of ISO 9001:2008, 4.1, in relation to the organizational processes as follows (see 5.4.2, and 7.4.1 for additional guidance on outsourcing).

a) Process identification and application

The organization should also identify the processes for software development, operation or maintenance.

b) Process sequence and interaction

The organization should also define the sequence and interaction of the processes in

1) life cycle models for software development, e.g. waterfall, incremental and evolutionary, and

2) quality and development planning, which should be based upon a life cycle model.

NOTE For further information, see the following:

— ISO/IEC 12207[11] and ISO/IEC 12207:1995/Amd.1:2002[12] (software life cycle processes) which define a set of software life cycle processes that may be used for reference;

— ISO/IEC TR 15271:1998[21], Annex C (guide to ISO/IEC 12207) which provides guidance on how to use processes from ISO/IEC 12207 in different life cycles.

Guideline Application

1. Description

A **quality management system** (QMS) can be expressed as the organizational structure, procedures, processes and resources needed to implement quality management.

Picture III-1: Quality Management System Structure

- **Quality Policies**

At the base of any organization's *ISO 9000* compliant QMS are the quality policies. The quality policies identify the main goals of the QMS and explain why an organization is adopting them. An organization's quality policy can be a very simple statement of its broad goals, or it can be a much more detailed document that defines the goals, objectives and responsible parties for each area that the *ISO 9000* standard impacts.

For a quality policy to conform to the standard it must be appropriate to the organization's purpose, communicated throughout the organization, reviewed for suitability, and continually improving its effectiveness and compliance to customer requirements.

The quality policy is a critical starting point for an effective *ISO 9000* implementation. It reflects a commitment from top management and is the vehicle for communicating objectives throughout the organization. While the statement may be brief, the consistency of the message should me maintained throughout the organization.

- **Quality Procedures**

The next level of documentation required of ISO compliant companies is the quality procedures document. Quality procedures need to address each element of the *ISO 9000* standard, and specify what processes are to be done, who in the organization will perform these processes, and what procedures will be followed for performing them.

ISO not only requires that an organization documents its processes and procedures, but also that it controls the

documentation in such a way that the following requirements are met:

- Current versions of the documentation must be readily available to all employees who perform the work being documented.

- Obsolete or superseded documentation must be removed from the system, or clearly identified so as to prevent their inadvertent use.

- All documentation, and any changes to it, must be reviewed and approved by functionally competent personnel.

- ISO documents are most effectively controlled and distributed using online electronic document delivery. If electronic document delivery is not available, hard copy documents must be maintained. All manuals must be promptly updated as changes are released. The number and severity of the non-conformances found will have a negative impact on the maintenance of ISO certification.

- **Work Instructions**

Work instructions are the lowest level of required procedural documentation for an ISO compliant quality management system. At their most basic level, work instructions explain, step by step, how a task is performed. Work instructions are not required for every task related to quality, rather only those which explain the manner of production, installation and servicing, and whose absence would adversely affect quality.

A common mistake for companies to make is to get bogged down in writing detailed work instructions. Such an approach is dangerous because it:

1) is time consuming,

2) it is not necessary to comply with the standard,

3) creates an unwieldy QMS, and

4) creates many opportunities for a registrar to document non-conformance with the QMS.

As you create work instructions, be prepared for there to be some areas of disagreement on the best way to perform certain tasks. The instruction writing process can be very helpful in reducing variation. For instance, if there are large differences in the way the first-shift and second-shift operators set up the same machine, establishing work instructions will help you solve that source of process variation. Work instructions should be flexible and change with the organization. From an ISO perspective, the

most important consideration is to make sure operators are working with an approved copy of the latest work instruction.

- **Quality Records**

In the *ISO 9000* standard, quality records are kept to demonstrate conformance to specified requirements and to the effective operation of the QMS. The ISO standard clearly mandates that certain things must be recorded and treated as *quality records*. These quality records can be considered the minimum required under any quality system that complies with the standard. Samples of quality records include part specifications, engineering changes, training course descriptions, product specifications, contract reviews, corrective action Requests, non-conformances and others.

When quality records are created as a consequence of a work instruction, the work instruction must list those quality records. A record, of any kind, is a document that furnishes objective evidence of activities performed or results achieved. Its nature is to capture something that has already happened. The specified requirements are those that apply to the product. The quality record must also demonstrate the effective operation of the QMS.

For example, a newly created test procedure is not a quality record, although generally it is a controlled document. It becomes a quality record when it is used to record the steps executed during a specific test to establish that the product met requirements. Similarly, a design drawing generally is not a quality record, although generally it is a controlled document. However, it becomes a quality record when it is part of a review package that establishes that the design was verified (i.e. evidence of effective operation of the QMS).

In both these cases, when the document in question (e.g. procedure, drawing) becomes a quality record, it loses its active nature and becomes fixed. The procedure or drawing that lives on to serve further purposes is not the same entity as the one that is set aside as a quality record. Similar logic can be applied to other instances.

From the standpoint of ISO audits, registrars may not be in a position to observe compliance with a specific quality procedure. As such, an authentic quality record may be the next best form of objective evidence. Any number of employees in an organization may generate quality records, which will need to be retrieved frequently for internal audits and registration audits.

The organization itself has the latitude (and responsibility) to determine which records are the most authentic and most comprehensive for the features that must be documented. For

instance, there may be two or three records that may document the satisfaction of a particular requirement. The organization may designate which of the multiple records should be retained as the primary quality record to prove satisfaction of that requirement. The supplemental records need not be retained as the official or designated proof.

- **Corrective Action Requests**

Corrective action is the cornerstone of continuous improvement, which is ultimately the goal of an *ISO 9000* compliant QMS. Corrective Action Requests (CAR), are controlled documents that reflect the actions taken to fix problems related to an organization's QMS. These problems can be non-conformances where the QMS does not measure up to the requirements of the standard, or more commonly, problems that have occurred as the organization produces its finished goods or services. The most common ways of uncovering these problems are through customer complaints, internal audits and quality control inspections.

The most effective means of resolving CARs is to determine how the CAR system initially broke down, as a properly completed CAR will not only identify the problem, but will also address a course of action which will help prevent the problem from reoccurring. When considering recommended changes to one particular problem, it may be beneficial to step back and take a look at the whole system and how changes will affect it.

2. Application

In this project, every part of a QMS (as organizational structure, procedures, processes and resources needed to implement a quality management system) will be applied as concrete guidelines for each part, due their importance.

Guideline Application Tool

1. Description

Because every part of a QMS will be applied as a concrete guideline, the applications tools will vary according to their type.

4.2. Documentation Requirements

4.2.1 General

ISO 9001:2008 Standard

The quality management system documentation shall include

a) documented statements of a quality policy and quality objectives,

b) a quality manual,

c) documented procedures and records required by this International Standard, and

d) documents, including records, determined by the organization to be necessary to ensure the effective planning, operation and control of its processes.

NOTE 1 Where the term "documented procedure" appears within this International Standard, this means that the procedure is established, documented, implemented and maintained. A single document may address the requirements for one or more procedures. A requirement for a documented procedure may be covered by more than one document.

NOTE 2 The extent of the quality management documentation can differ from one organization to another due to

a) the size of organization and type of activities

b) the complexity of processes and their interactions, and

c) the competence of personnel.

NOTE 3 The documentation can be in any form or type of medium.

ISO 90003:2004 Guideline

Documents for the effective planning, operation, and control of processes for software may cover the following:

1) descriptions of processes, such as those identified in implementing 4.1;

2) descriptions of procedural instructions and/or templates used;

3) descriptions of life cycle models used, such as waterfall, incremental and evolutionary;

4) descriptions of tools, techniques, technologies, and methods such as those identified in implementing 4.1;

5) technical topics such as standards or guidance documents for coding, design and development, and testing.

NOTE For further information on document identification as part of configuration management, see 7.5.3.

Guideline Application

1. Description

In this guideline I will describe some tools, frameworks, software life cycles and guidance documents used in this project.

4.2.1.1. Ubuntu GNU/Linux

1. Description

Ubuntu is a computer operating system originally based on the *Debian* distribution and distributed as free and open source software with additional proprietary software available. Provides an up-to-date, stable operating system for the average user, with a strong focus on usability and ease of installation.

Ubuntu is composed of many software packages, of which the vast majority are distributed under a free software license. The main license used is the GNU General Public License (GNU GPL) which, explicitly declares that users are free to run, copy, distribute, study, change, develop and improve the software.

Ubuntu is sponsored by the UK-based company Canonical Ltd. By keeping *Ubuntu* free and open source, Canonical is able to utilize the talents of community developers in Ubuntu's constituent components.

I will use *Ubuntu 9.04* released in April 2009 as operating system. More information can be found in http://www.ubuntu.com/.

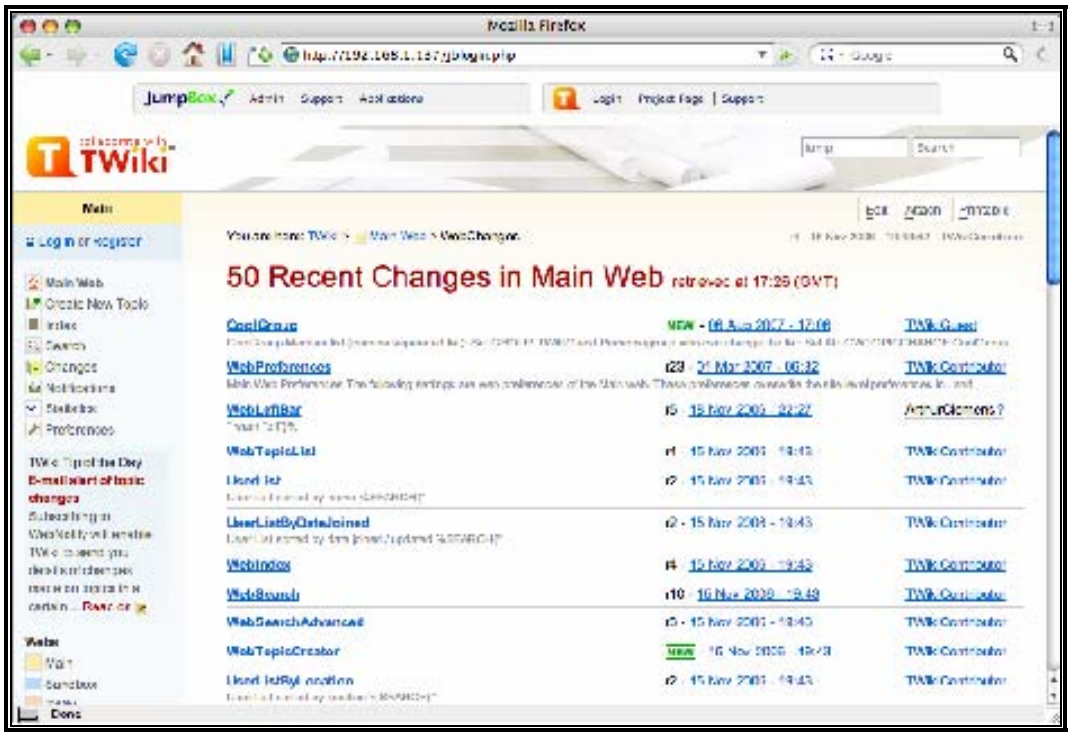
4.2.1.2. Twiki

1. Description

TWiki is a flexible, powerful, and easy to use enterprise wiki, enterprise collaboration platform, and web application platform. It is a Structured Wiki, typically used to run a project development space, a document management system, a knowledge base, or any other groupware tool, on an intranet, extranet or the Internet. Users without programming skills can create web applications.

Developers can extend its functionality with plugins. *TWiki* fosters information flow within an organization, lets distributed teams work together seamlessly and productively, and eliminates the one-webmaster syndrome of outdated intranet content. *TWiki* has been downloaded over 500,000 times and is used daily by millions of people in over 100 countries. Some larger deployments have several 100,000 pages and over 10,000 users.

Picture III-2: Twiki Welcome Main Page



Picture III-3: Twiki Web Changes Page

I will use *Twiki* as an enterprise collaboration suit. More information can be found in http://twiki.org/.

4.2.1.3. Volere Requirements Model

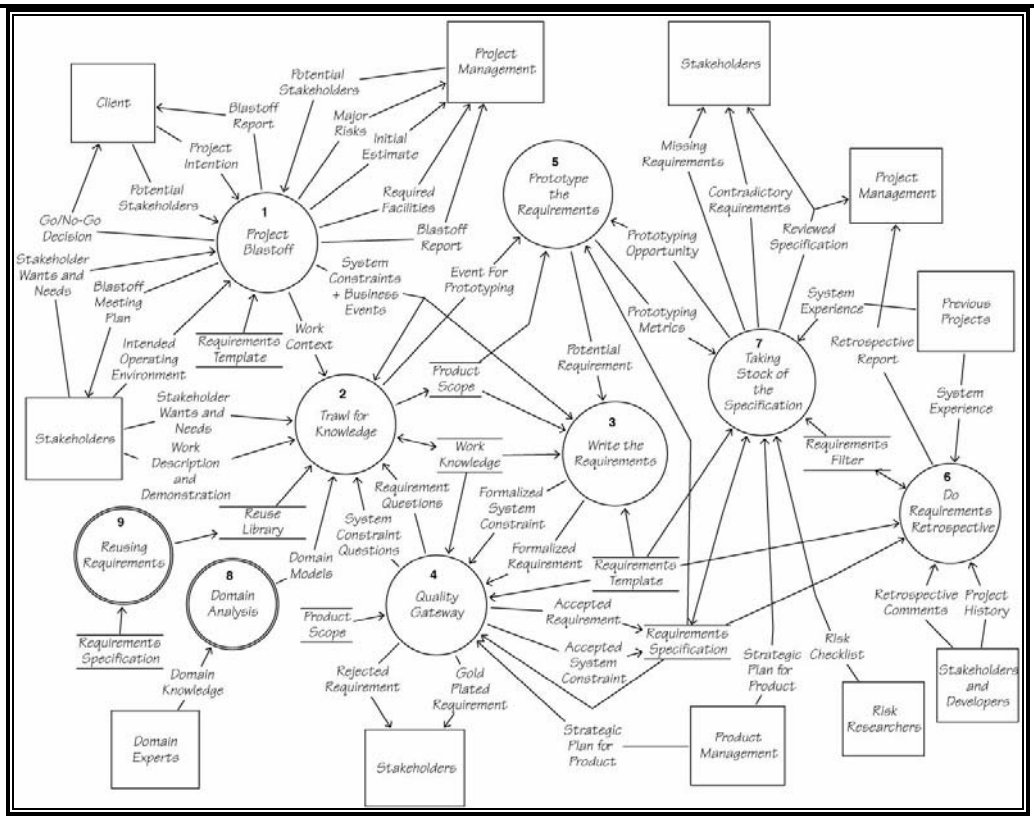
1. Description

The most useful products are those where the developers have understood what the product is intended to accomplish for its users and how it must accomplish that purpose. To understand them, you must understand what kind of work the users want to do and how the product will affect that work and fit into the organization's goals. What the product does for its users and which constraints it must satisfy in this context are the product's requirements.

Volere requirement process is a ser of principles, processes, templates, tools, and techniques developed to improve the discovery, communication, and management of requirements.

2. Features

This requirements process model is organized as a hierarchy of processes. This is a model of the generic processes necessary to elicit, specify, and review requirements. The model focuses on content. The dependencies between the processes are defined by the interfaces. This model does not imply any sequence, but rather is an asynchronous network. Any combination of processes can be active at any time.



Picture III-4: Volere Requirements Process Model Summary

I will user the *Volere requirement process* for the requirements elicitation process. To know more see Suzanne and James Robertson or visit http://www.volere.co.uk/.

4.2.1.4. Jmeter

1. Description

JMeter is a 100% pure Java desktop application designed to load test functional behaviour and measure performance. It was originally designed for testing Web Applications but has since expanded to other test functions.

2. Features

JMeter may be used to test performance both on static and dynamic resources (files, Servlets, Java Objects, Data Bases and Queries, FTP Servers). It can be used to simulate a heavy load on a server, network or object to test its strength or to analyze overall performance under different load types. You can use it to make a graphical analysis of performance or to test your server/script/object behaviour under heavy concurrent load.

JMeter can load and performance test many different server types: Web - HTTP, HTTPS, SOAP, Database via JDBC, LDAP, JMS Mail - POP3.

I will use *JMeter* to as the integration test tool. More information can be found in http://jakarta.apache.org/jmeter/.

4.2.1.5. Maven

1. Description

Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), *Maven* can manage a project's build, reporting and documentation from a central piece of information.

2. Features

Maven's primary goal is to allow a developer to comprehend the complete state of a development effort in the shortest period of time. In order to attain this goal there are several areas of concern that Maven attempts to deal with:

- **Making the build process easy:** While using *Maven* doesn't eliminate the need to know about the underlying mechanisms, *Maven* does provide a lot of shielding from the details.
- **Providing a uniform build system:** *Maven* allows a project to build using its project object model and a set of plugins that are shared by all projects using Maven, providing a uniform build system. Once you familiarize yourself with how one *Maven* project builds you automatically know how all Maven projects build saving you immense amounts of time when trying to navigate many projects.
- **Providing quality project information:** *Maven* provides plenty of useful project information that is in part taken from your POM and in part generated from your project's sources.
- **Providing guidelines for best practices development:** *Maven* aims to gather current principles for best practices development, and make it easy to guide a project in that direction. For example, specification, execution, and reporting of unit tests are part of the normal build cycle using Maven. *Maven* also aims to assist in project workflow such as release management and issue tracking. *Maven* also suggests some guidelines on how to layout your project's directory structure so that once you learn the layout you can easily navigate any other project that uses *Maven* and the same defaults.

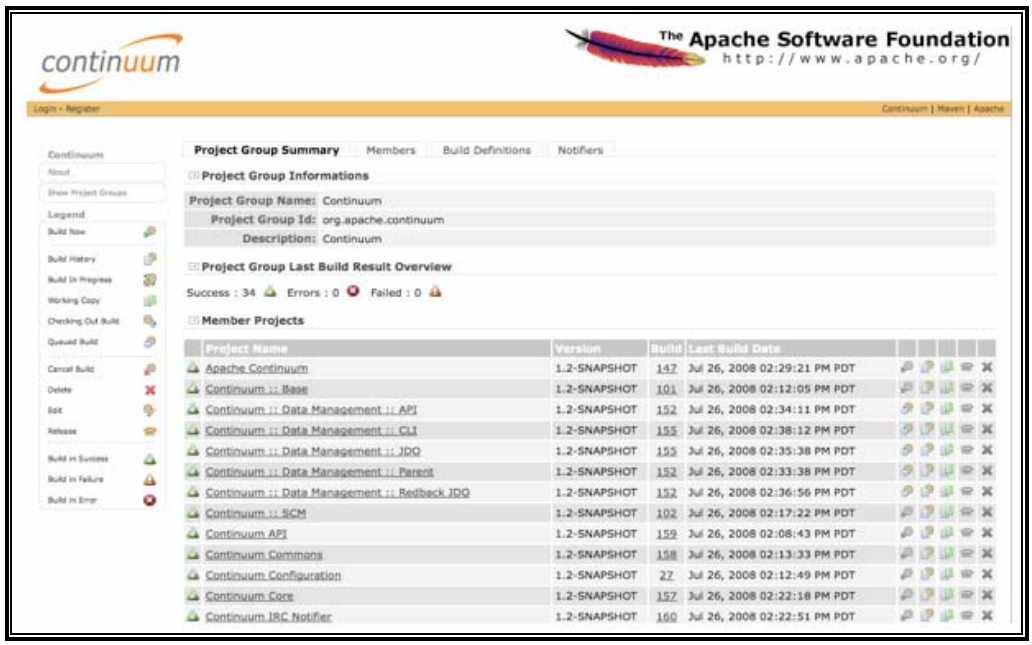
• **Allowing transparent migration to new features:** *Maven* provides an easy way for *Maven* clients to update their installations so that they can take advantage of any changes that been made to *Maven* itself.

I will use *Maven* as project management and comprehension tool. More information can be found in http://maven.apache.org.

4.2.1.6. Continuum

1. Description

Continuum is an enterprise-ready continuous integration server with features such as automated builds, release management, role-based security, and integration with popular build tools and source control management systems. Whether you have a centralized build team or want to put control of releases in the hands of developers, *Continuum* can help you improve quality and maintain a consistent build environment.



Picture III-5: Continuum Snapshoot

2. Features

• **Easy installation:** Download the standalone application and run it or deploy the *Continuum* war in your servlet container.

• **Easy Configuration:** Projects builds are auto-configured but they can be configured easily with the web interface.

• **SCM support:** *CVS*, *Subversion*, *Clear case*, *Perforce*… are supported.

• **Change set support:** For each build result, Continuum print all scm changes (commit authors, commit logs, modified files).

• **Build notification:** *Mail*, *Jabber* and *Google Talk*, *MSN*, *IRC*, report deployment with *wagon*.

• **Build tool support:** *Maven* 1 and 2, *ANT*, shell scripts.

• **External Access:** External tools can interact with *Continuum* with *XMLRPC* API.

• **Build type:** Manual, scheduled and push (with xmlrpc) build technique are supported.

• **Build template:** Users can define default build templates to use by project type.

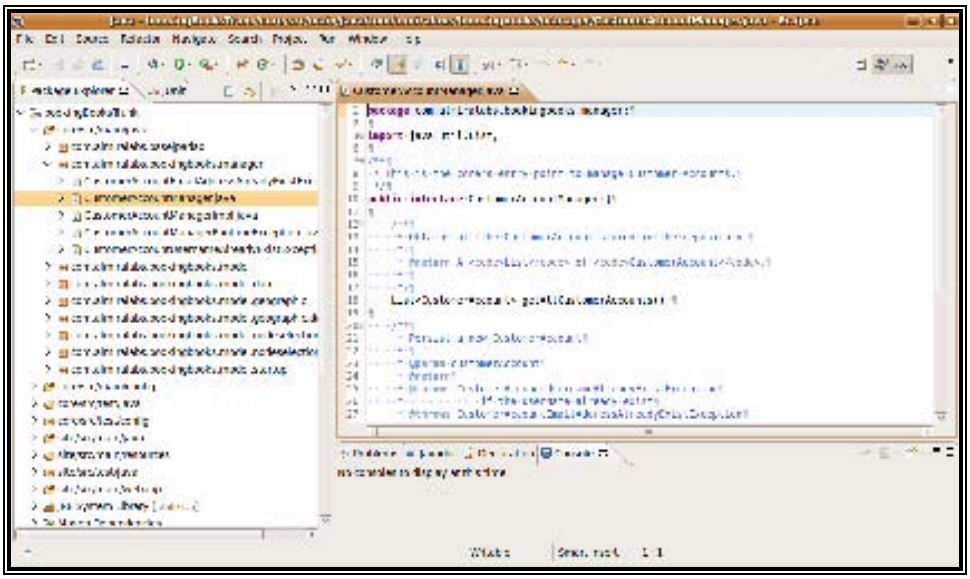
• **Build queue:** Users can view all projects in the queue and interrupt some builds.

I will use *Continuum* as continuous integration server. More information can be found in http://continuum.apache.org.

4.2.1.7. Eclipse

1. Description

Eclipse is a multi-language software development environment comprising an IDE and a plug-in system to extend it. It is written primarily in *Java* and can be used to develop applications in *Java* and, by means of the various plugins, in other languages as well.



Picture III-6: Eclipse Snapshoot

2. Features

The *Eclipse* open source community has over 60 open source projects. These projects can be conceptually organized into seven different "pillars" or categories:

- Enterprise Development
- Embedded and Device Development
- Rich Client Platform
- Rich Internet Applications
- Application Frameworks
- Application Lifecycle Management (ALM)
- Service Oriented Architecture (SOA)

I will use *Eclipse* as my software development environment. More information can be found in http://www.eclipse.org.

4.2.1.8. JBoss Application Server

1. Description

JBoss application server is a free software/open-source Java EE-based application server. Because it is Java-based, the *JBoss* application server operates cross-platform: usable on any operating system that *Java* supports.

2. Features

JBoss Application Server is a Java EE certified platform for developing and deploying enterprise *Java* applications, Web applications, and Portals. *JBoss* application server provides the full range of *Java EE* features as well as extended enterprise services including clustering, caching, and persistence.

I will use *JBoss* as my application server. More information can be found in http://jboss.org/jbossas.

4.2.1.9. Jira

1. Description

JIRA is a proprietary enterprise software product, commonly used for bug tracking, issue tracking, and project management. *JIRA* is used as a way to manage bug tracking for large-scale open source and public projects.

I will use Jira as issue tracking and project management tool. More information can be found in http://www.atlassian.com/software/jira/.

4.2.1.10. Spring

1. Description

The *Spring* framework is an open source application framework for the *Java* platform and *.NET Framework*.

The first version was written by Rod Johnson who released the framework with the publication of his book Expert One-on-One J2EE Design and Development. The core features of *Spring* can be used by any *Java* application, but there are extensions for building web applications on top of the *Java EE* platform.

2. Features

Central to the *Spring* is its inversion of control container (IOC), which provides a consistent means of configuring and managing *Java* objects using call backs. The container is responsible for managing object lifecycles: creating objects, calling initialization methods, and configuring objects by wiring them together.

Objects created by the container are also called Managed Objects or Beans. Typically, the container is configured by loading XML files containing Bean definitions which provide the information required to create the beans.

Objects can be obtained by means of dependency lookup or dependency injection. Dependency lookup is a pattern where a caller asks the container object for an object with a specific name or of a specific type. Dependency injection is a pattern where the container passes objects by name to other objects, via constructors, properties, or factory methods.

In many cases is not necessary to use the container when using other parts of the *Spring* framework, although using it will likely make an application easier to configure and customize. The *Spring* container provides a consistent mechanism to configure applications and integrates with almost all *Java* environments, from small-scale applications to large enterprise applications.

I will use *Spring* to keep my web application with high cohesion and low coupling. More information can be found in http://www.springsource.org/.

4.2.1.11. JPA

1. Description

The *Java Persistence API*, sometimes referred to as *JPA*, is a Java programming language framework managing relational data in

applications using Java Platform Standard Edition and Java Platform Enterprise Edition.

2. Features

Java Persistence API (*JPA*) provides POJO (Plain Old Java Object) standard and object relational mapping (OR mapping) for data persistence among applications. Persistence, which deals with storing and retrieving of application data, can now be programmed with *Java Persistence API* starting from *EJB* 3.0 as a result of JSR 220. This API has borrowed many of the concepts and standards from leading persistence frameworks like *Toplink* (from Oracle) and *Hibernate* (from JBoss).

I will use *JPA* as the API to persist *Java Objects*. More information can be found in: http://www.oracle.com/technetwork/articles/javaee/jpa-137156.html/.

4.2.1.12. Hibernate

1. Description

Hibernate is a powerful, high performance object/relational persistence and query service.

2. Features

Hibernate lets you develop persistent classes following object-oriented idiom - including association, inheritance, polymorphism, composition, and collections. *Hibernate* allows you to express queries in its own portable SQL extension (HQL), as well as in native SQL, or with an object-oriented Criteria and Example API.

The LGPL open source license allows the use of *Hibernate* and in open source and commercial projects.

I will use *Hibernate* to persist *Java Objects* via Object/Relational mapping. More information can be found in: http://www.hibernate.org/.

4.2.1.13. JavaServer Pages

1. Description

JavaServer Pages (*JSP*) technology enables Web developers and designers to rapidly develop and easily maintain, information-rich, dynamic Web pages that leverage existing business systems.

As part of the Java technology family, *JSP* technology enables rapid development of Web-based applications that are platform independent. *JSP* technology separates the user interface from

content generation, enabling designers to change the overall page layout without altering the underlying dynamic content.

2. Features

If you are a Web page developer or designer who is familiar with HTML, you can:

- Use *JSP* technology without having to learn the Java language: You can use JSP technology without learning how to write Java scriplets. Although scriptlets are no longer required to generate dynamic content, they are still supported to provide backward compatibility.

- Extend the *JSP* language: Java tag library developers and designers can extend the *JSP* language with "simple tag handlers," which utilize a new, much simpler and cleaner, tag extension API. This spurs the growing number of pluggable, reusable tag libraries available, which in turn reduces the amount of code needed to write powerful Web applications.

- Easily write and maintain pages: The *JavaServer Pages Standard Tag Library* (*JSTL*) expression language is now integrated into *JSP* technology and has been upgraded to support functions. The expression language can now be used instead of scriptlet expressions.

I will use *JSP* to develop the dynamic Web Page (the View in the Model-View-Controled design pattern). More information can be found in: http://java.sun.com/products/jsp/.

4.2.1.14. JUnit

1. Description

JUnit is a simple framework to write repeatable tests. It is an instance of the xUnit architecture for unit testing frameworks.

I will use *JUnit* to write repeteable tests. More information can be found in: http://junit.sourceforge.net/.

4.2.1.15. EasyMock

1. Description

EasyMock provides Mock Objects for interfaces (and objects through the class extension) by generating them on the fly using Java's proxy mechanism. Due to *EasyMock's* unique style of recording expectations, most refactorings will not affect the mock objects. So *EasyMock* is a perfect fit for test-driven development.

2. Features

Unit testing is the testing of software units in isolation. However, most units do not work alone, but they collaborate with other units. To test a unit in isolation, we have to simulate the collaborators in the test.

A Mock Object is a test-oriented replacement for a collaborator. It is configured to simulate the object that it replaces in a simple way. In contrast to a stub, a mock object also verifies whether it is used as expected.

I will use *EasyMock* as white box testing tool for unitary test. More information can be found in http://easymock.org/.

4.2.1.16 Xuse

1. Description

The *Xuse* project aims to define an XML data model that will allow for the capture and management of software requirements from a use-case centric perspective. The project will provide XSLTs to create multiple views on this data model in a variety of formats such as HTML or PDF.

2. Features

As all the data will be stored in XML it will also be possible to provide different information to different consumers: for example you may wish to document technical issues of why a software requirement will be difficult to implement but not show this information to the customer who is signing off the requirements. Additionally different people such as QA engineers and developers may require different views on the data - something that XSLT facilitates. It is also hoped that users of Xuse will be free to create their own custom views on the data simply by providing a new XSLT.

I will use *Xuse* to capture and manage all the requirements and use cases. More information can be found in: http://xuse.sourceforge.net/.

4.2.1.17 Subversion

1. Description

Subversion is a full-featured version control system. *Subversion* has since expanded beyond its original goal of replacing

CVS, but its basic model, design, and interface remain heavily influenced by that goal.

2. Features

CVS is a relatively basic version control system. For the most part, *Subversion* has matched or exceeded *CVS's* feature set where those features continue to apply in *Subversion's* particular design.

• Directories are versioned: *Subversion* versions directories as first-class objects, just like files.

• Copying, deleting, and renaming are versioned: copying and deleting are versioned operations. Renaming is also a versioned operation.

• Atomic commits: no part of a commit takes effect until the entire commit has succeeded. Revision numbers are per-commit, not per-file, and commits log message is attached to its revision, not stored redundantly in all the files affected by that commit.

• Branching and tagging are cheap (constant time) operations. Branches and tags are both implemented in terms of an underlying copy operation. A copy takes up a small, constant amount of space. Any copy is a tag; and if you start committing on a copy, then it's a branch as well.

• Merge tracking: automated assistance with managing the flow of changes between lines of development, and with the merging of branches back into their sources.

• File locking: *Subversion* supports (but does not require) locking files so that users can be warned when multiple people try to edit the same file.

• Standalone server option (svnserve): offers a standalone server option using a custom protocol, since not everyone wants to run an Apache HTTPD server.

• Parseable output: all output of the command-line client is carefully designed to be both human readable and automatically parseable; scriptability is a high priority.

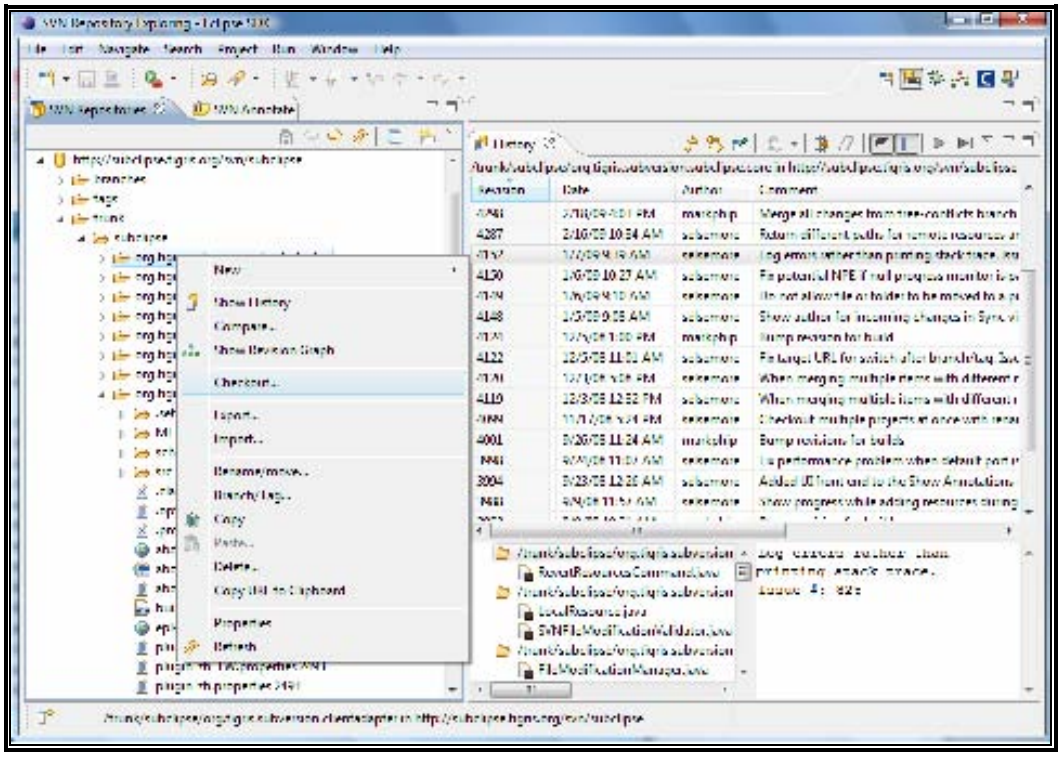
• Interactive conflict resolution: the command-line client (svn) offers various ways to resolve conflicting changes, include interactive resolution prompting.

• Repository read-only mirroring: *Subversion* supplies a utility, svnsync for synchronizing (via either push or pull) a read-only slave repository with a master repository.

• Binary files handled efficiently: *Subversion* is equally efficient on binary as on text files, because it uses a binary diffing algorithm to transmit and store successive revisions.

• Costs are proportional to change size, not data size: in general, the time required for a Subversion operation is proportional to the size of the changes resulting from that operation, not to the absolute size of the project in which the changes are taking place.

• Changelists: allows a user to put modified files into named groups on the client side, and then commit by specifying a particular group.



Picture III-7: Subversion Eclipse Plugin Snapshoot

I will use *Subversion* as version control system. More information can be found in http://subversion.apache.org/.

4.2.1.18. PostgreSQL

1. Description

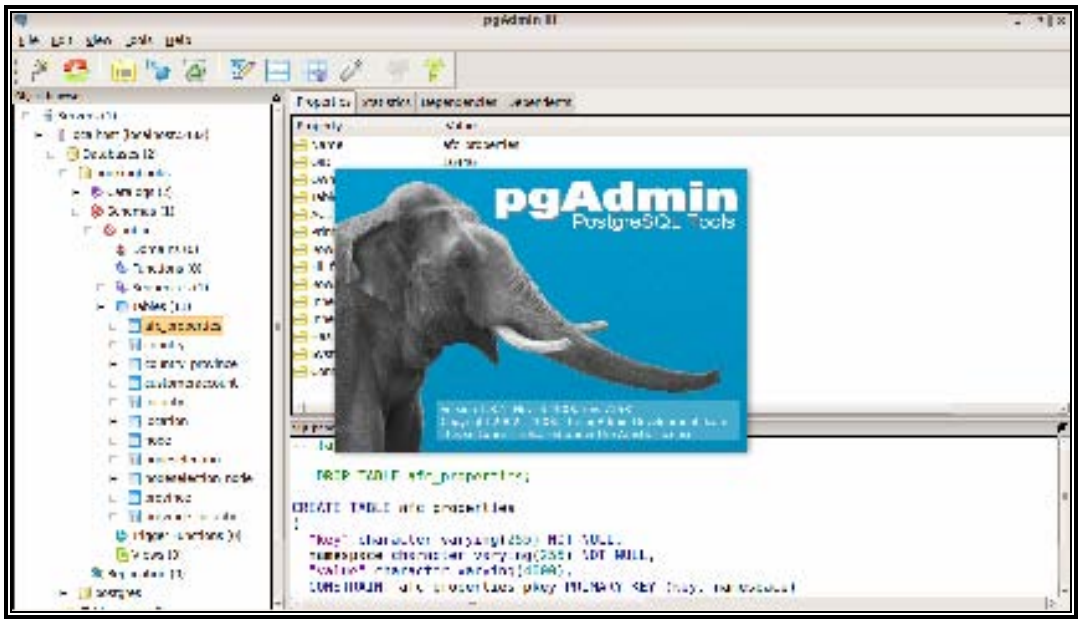
PostgreSQL is a powerful, open source object-relational database system. It has more than 15 years of active development and a proven architecture that has earned it a strong reputation for reliability, data integrity, and correctness.

2. Features

PostgreSQL has full support for foreign keys, joins, views, triggers, and stored procedures (in multiple languages). It includes most SQL:2008 data types, including INTEGER, NUMERIC,

BOOLEAN, CHAR, VARCHAR, DATE, INTERVAL, and TIMESTAMP. It also supports storage of binary large objects, including pictures, sounds, or video.

An enterprise class database, *PostgreSQL* boasts sophisticated features such as Multi-Version Concurrency Control, point in time recovery, tablespaces, asynchronous replication, nested transactions (savepoints), online/hot backups, a sophisticated query planner/optimizer, and write ahead logging for fault tolerance. It supports international character sets, multibyte character encodings, Unicode, and it is locale-aware for sorting, case-sensitivity, and formatting. It is highly scalable both in the sheer quantity of data it can manage and in the number of concurrent users it can accommodate.



Picture III-8: PosgtreSQL pgAdmin tool

I will use *PostgreSQL* as database system and pgAdmin as administration and development platform for *PostgreSQL*. More information can be found in http://www.postgresql.org/ and in http://www.pgadmin.org/.

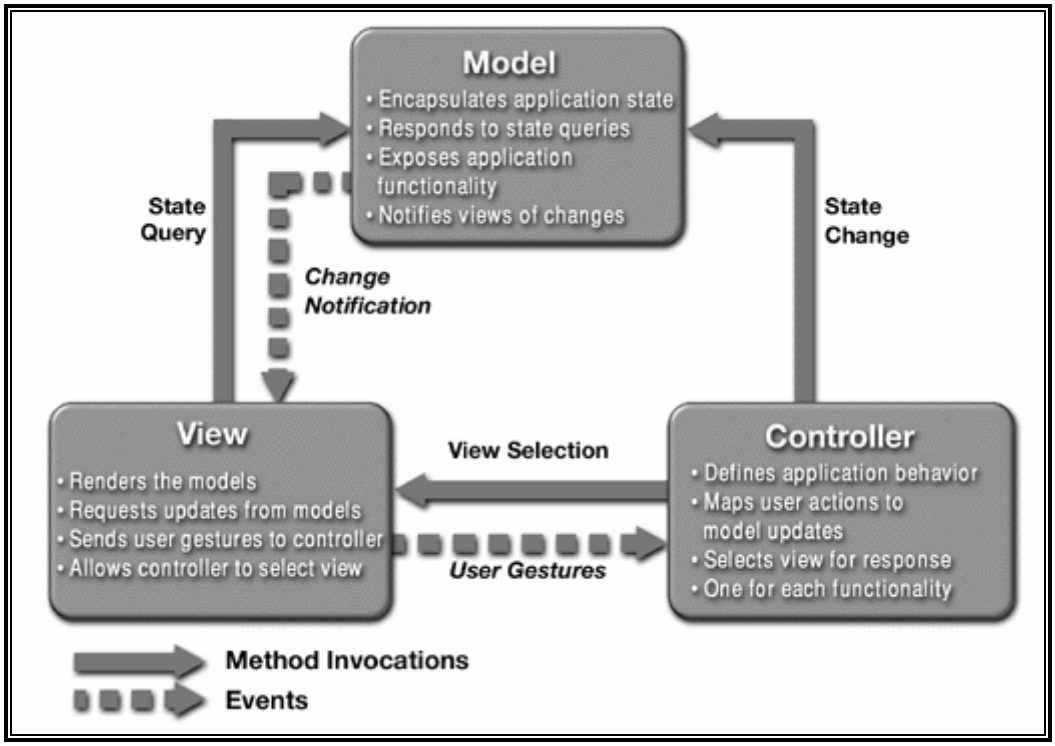
4.2.1.19. Software Macro Architecture

1. Description

The software architecture used in this project is based in the Model-View-Controller (MVC) Pattern.

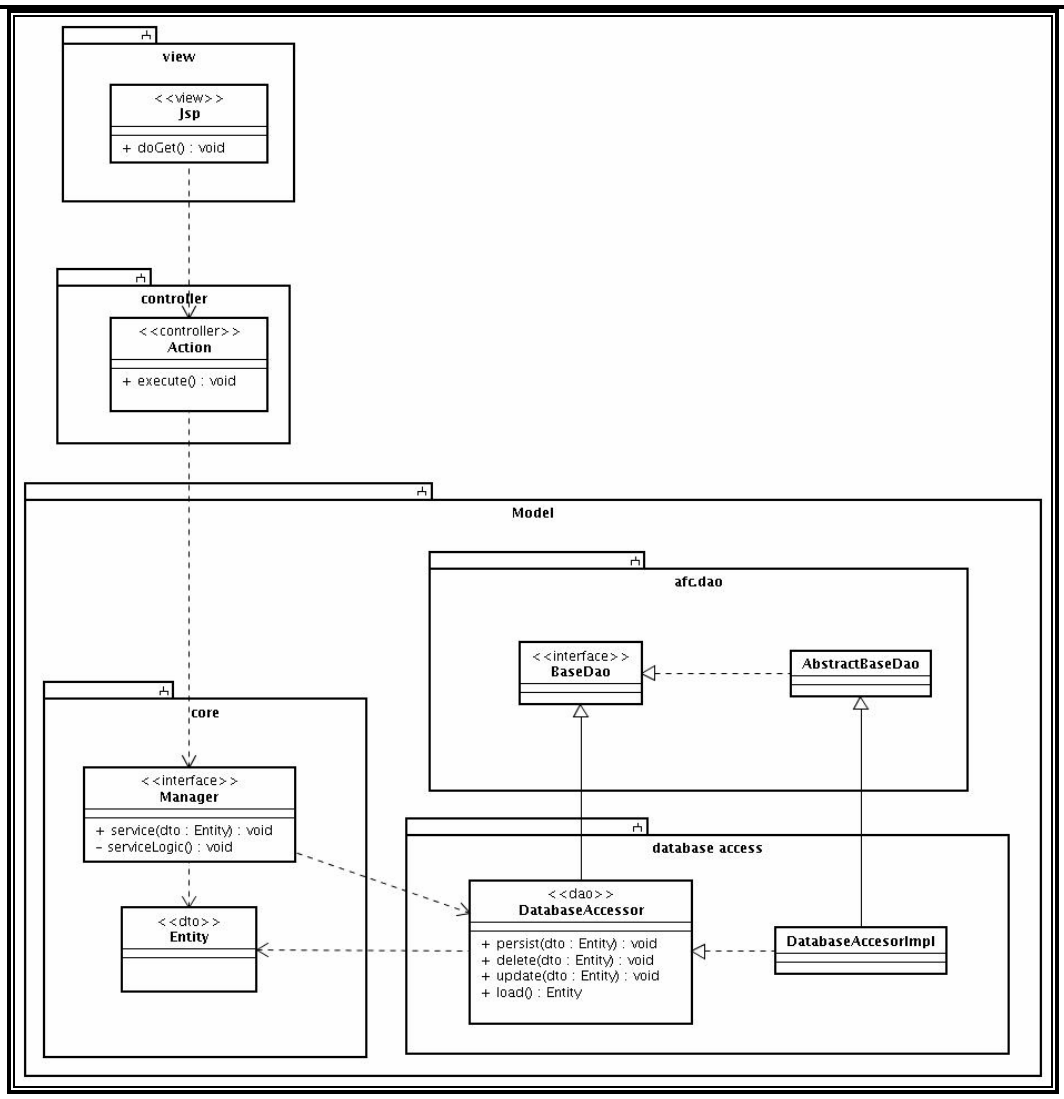
MVC is a classic design pattern often used by applications that need the ability to maintain multiple views of the same data. The MVC pattern hinges on a clean separation of objects into one of

three categories: models for maintaining data, views for displaying all or a portion of the data, and controllers for handling events that affect the model or views.



Picture III-9: Model View Controller Overview Diagram

Because of this separation, multiple views and controllers can interface with the same model. Even new types of views and controllers that never existed before can interface with a model without forcing a change in the model design.



Picture III-10: Model View Controller Class Diagram

2. Features

- **View:** the View in MVC architecture represents the page design code. It has been done using the *Java Server Pages* (JSP) technology. *JSP* technology provides a simplified, fast way to create dynamic web content. In the class diagram, the view has a method *doGet()* to obtain some values from the controller.

- **Controller:** the Controller in the *MVC* architecture represents the navigational code. It must handle the request provided by the view, and create a response (handler that transfers control to another resource). I have selected *Struts* as Controller. This controller has the method *execute()*, which communicates the Model (which represents the business login) with the view.

- **Model:** the Model in the MVC architecture, represents the business or database code. In this diagram, the model is composed of:

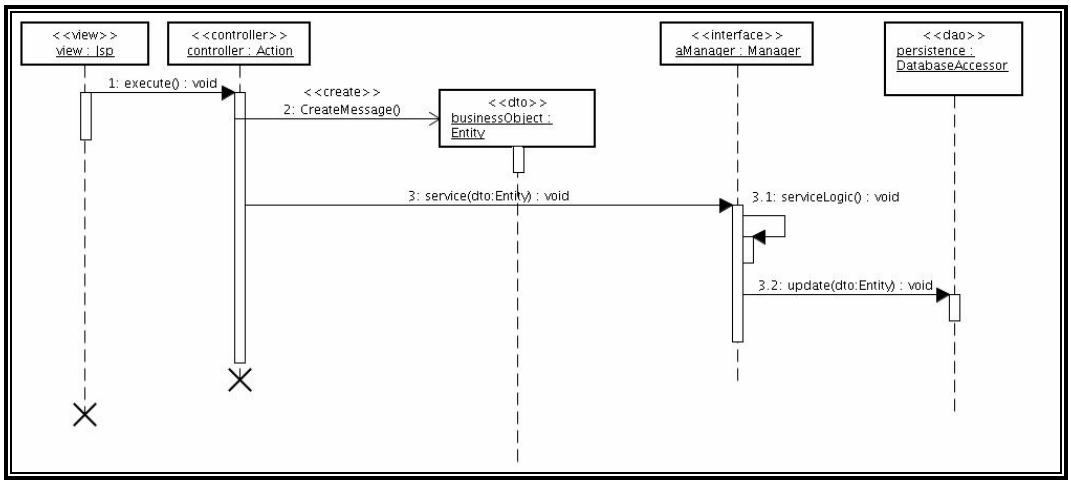
- Manager: is an interface that contains the functionality of the use case. Represents the business logic of the application, it will be used by the controller to perform the operations described in the use case.

- Entity: is a class that represents the data that will be saved in a data base. For example, in the entity Customer, a possible attributes should be Username, Password or Email Address. I have used *JPA/Hibernate* as Object-relational-mapping to map a *Java Object* to a relational database. This can be done adding some attributes to the elements defined in the entity.

- DAO elements: data access object (DAO) is an object that provides an interface to some type of database or persistence mechanism, providing some specific operations without exposing details of the database. It provides a mapping from application calls to the persistence layer. This isolation separates the concerns of what data accesses the application needs, in terms of domain-specific objects and data types (the public interface of the DAO), and how these needs can be satisfied with a specific DBMS, database schema, etc. (the implementation of the DAO).

Usually, the DAO elements contains the basic CRUD (Create, Read, Update, Delete) operations over an Entity (which are in the BaseDao interface), other basic operations (finders methods to search by a search criteria, for example search all customers with a username or a password) and operations related to the business logic.

- DTO elements: data type object (DTO) is a classification identifying one of various types of data, as floating-point, integer, or Boolean, stating the possible values for that type, the operations that can be done on that type, and the way the values of that type are stored.



Picture III-11: Model View Controller Interaction Diagram

4.2.1.20. Software Micro Architecture

1. Description

This micro architecture is influenced by the Unified *Software Development Process*; *USDP* is a software development process directed by use cases, focused in architecture, with an iterative and incremental approach. The use cases define this micro architecture because they are used from the stage of analysis to the final stage of test.

The main goal of this micro architecture is to define a set of rules to create the design and implementation from the use cases (David Lopez, 2006). Using these rules, we will generate software artefacts from the use cases.

2. Features (Rules)

2.1. For Each Use Case Create a Manager Facade with the Behaviour of the Actor

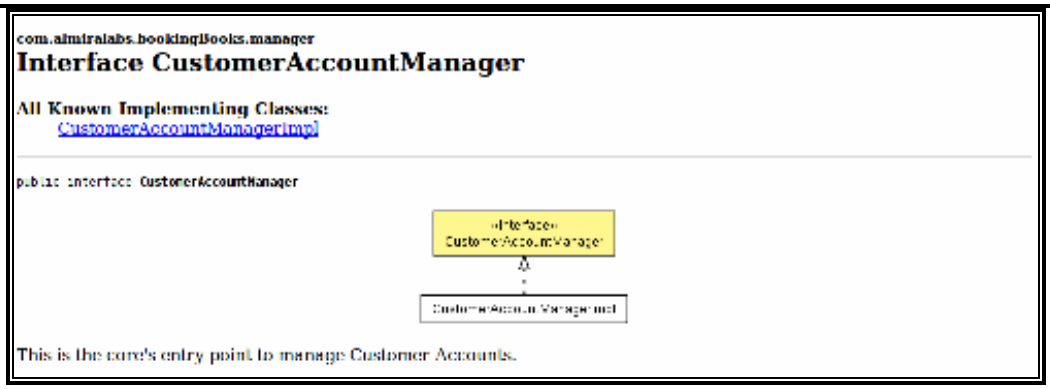
- **Purpose:** To provide an interface that contains the functionality of the use case, with the methods needed by the actor to achieve his goal.

- **Motivation:** A use case represents the functionality required by an actor to achieve a certain goal. Architecturally, this can be represented by a class that provides all the functionality needed to carry out the purpose of the Actor.

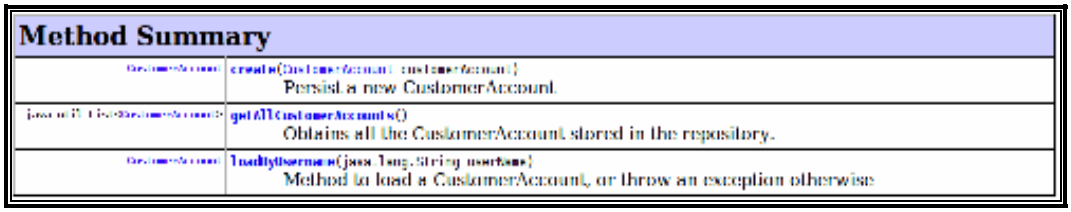
This class encapsulates all the functionality of the use case following the *Facade* design pattern, providing a simplified interface that encapsulates a large amount of functionality and access colleagues. The functionality is offered by functions (or methods).

- **Structure:** Create, for each use case, a class named as the u*se case name* and the suffix *Manager*. For example, if the use case is *Customer Account* the name of the class should be *CustomerAccountManager*.

This class is responsible for maintaining the conversational state actor - System, so using the *J2EE* conventions we will say that it is a stateful object. The function name and parameters will be derived from the action performed by the actor on the system. For example, if there is a flow of the use case named "the actor searches all customer accounts" we should create a method named *getAllCustomerAccounts()*.



Picture III-12: CustomerAccountManager Class Diagram



Picture III-13: CustomerAccountManager Method Summary

2.2. Throw Checked Exceptions in Managers if the Action is a Use Case Alternative Flow

- **Purpose:** To provide a mechanism to report an alternative flow of a use case due to the user actions.

- **Motivation:** It is a bad practice to return null objects or status codes to report the outcome of a particular transaction; instead of this approach, everything has to return the correct result (or nothing if the function returns no object), and if there is some error, throw an exception.

In Java there are two types of exceptions: checked (extends *java.lang.Exception*) which necessarily have to be treated by the caller object, and unchecked (extends *java.lang.RuntimeException* or extends *java.lang.Error*) which not have to be required to treated. We create checked exceptions to encapsulate all the use case alternative flows.

- **Structure:** if as a result of a user action, is executed an alternative flow, the function (or method) that implements this alternative flow must throw a checked exception type for each one of them.

For example, in Picture III-14: CustomerAccount ManagerImpl Class Diagram, the method *create (CustomerAccount customerAccount)*, throws the exceptions *CustomerAccountUsernameAlreadyExistException()* and *CustomerAccountEmailAddressAlreadyExistException()*. Those

exceptions have been done because in the use case there are two alternatives flows (see *Picture III-50: Use Case 001: Customer Registration*)

2.3. Throw Unchecked Exceptions in the Managers for Errors not Reflected in the Use Case

- **Purpose:** Define unrecoverable errors which are not written in the use case.

- **Motivation:** There are errors that are not defined in the use case as alternative flows, either functionally or because of no interest, or because they are dependent of the technology. A typical example is when using a library, for example the JDBC API. If for example there is a connection error with the database, the API throws a checked exception type. If not defined an alternative flow system how to behave to an error of this kind, we must transform this exception in an unchecked rate.

- **Structure:** convert checked exceptions into unchecked exceptions.

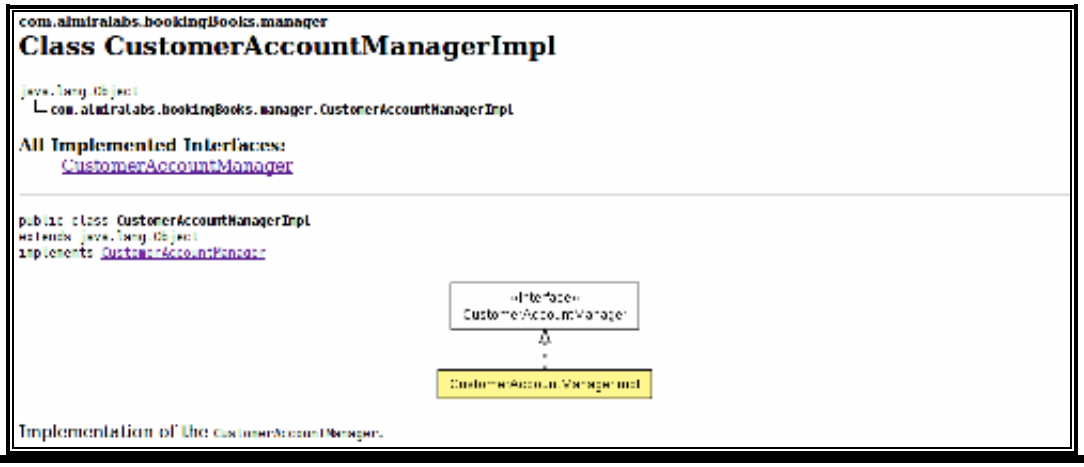
2.4. Implement the Manager Facades with its Business Logic

- **Purpose:** Implement the functionality described in the use case.

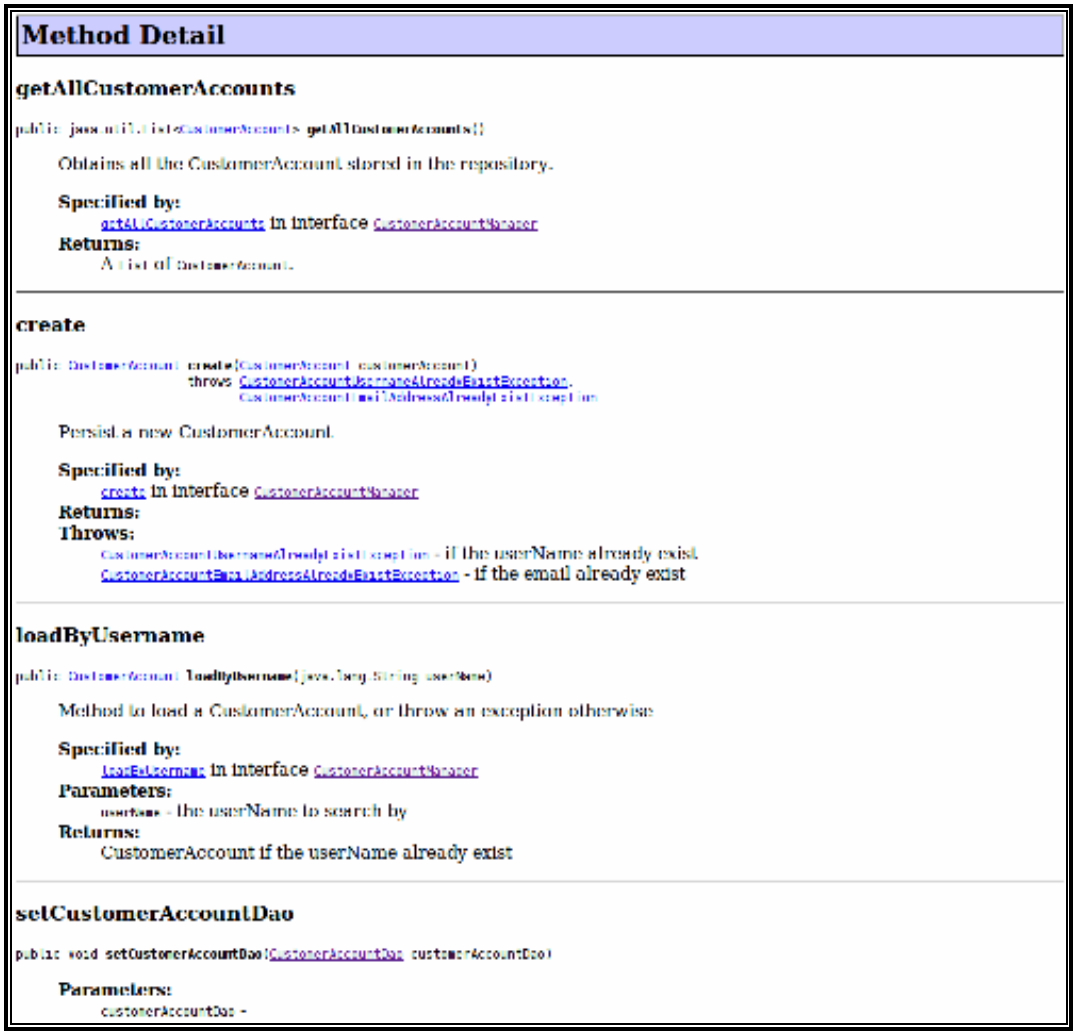
- **Motivation:** The business logic represents the functionality of the use case. It's not something that is part of architecture, but is simply code that checks and executes the actions of a use case. Business logic would interact with other components in the system which is developing, as our business objects and DAOs.

- **Structure:** write the business logic in methods (services) of the facade.

Obviously, must be done unitary test over these artifacts.



Picture III-14: CustomerAccount ManagerImpl Class Diagram



Method Detail

getAllCustomerAccounts

public java.util.List<CustomerAccount> getAllCustomerAccounts()

Obtains all the CustomerAccount stored in the repository.

Specified by:
getAllCustomerAccounts in interface CustomerAccountManager
Returns:
A list of CustomerAccount.

create

public CustomerAccount create(CustomerAccount customerAccount)
throws CustomerAccountUsernameAlreadyExistException,
CustomerAccountEmailAddressAlreadyExistException

Persist a new CustomerAccount

Specified by:
create in interface CustomerAccountManager
Returns:
Throws:
CustomerAccountUsernameAlreadyExistException - if the userName already exist
CustomerAccountEmailAddressAlreadyExistException - if the email already exist

loadByUsername

public CustomerAccount loadByUsername(java.lang.String userName)

Method to load a CustomerAccount, or throw an exception otherwise

Specified by:
loadByUsername in interface CustomerAccountManager
Parameters:
userName - the userName to search by
Returns:
CustomerAccount if the userName already exist

setCustomerAccountDao

public void setCustomerAccountDao(CustomerAccountDao customerAccountDao)

Parameters:
customerAccountDao -

Picture III-15: CustomerAccountManagerImpl Method Summary

2.5. Create Dao's Facades for each Business Object to Persist

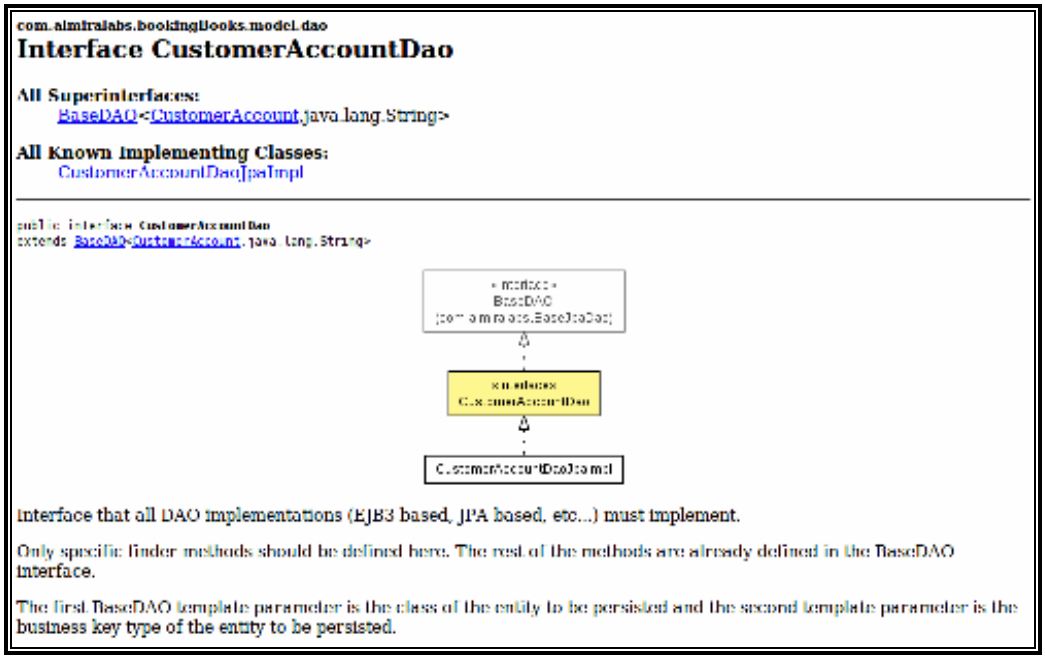
- **Purpose:** a **data access object** (DAO) is an object that provides an abstract interface to some type of database or persistence mechanism, providing some specific operations without exposing details of the database. It provides a mapping from application calls to the persistence layer. This isolation separates the concerns of what data accesses the application needs, in terms of domain-specific objects and data types (the public interface of the DAO), and how these needs can be satisfied with a specific DBMS, database schema, etc. (the implementation of the DAO).

- **Motivation:** The access code to an object is generally used by different parts of the system; therefore it is very reusable. With

the DAO component we can centralize all services of the business object persistence.

When we talk about persistence, we speak of a repository outside of the system, but not necessarily database. It may be a file system or a remote service accessible via **web service** or any other protocol.

● **Structure:** A DAO has CRUD (Create, Read, Update and Delete) methods, and additionally will have search methods (Finders) by if you need to approach the case.



Picture III-16: CustomerAccountDao Class Diagram

2.6. Add Exceptions to the DAO Interfaces

• **Purpose:** To establish a common behaviour of DAO components for exceptional cases.

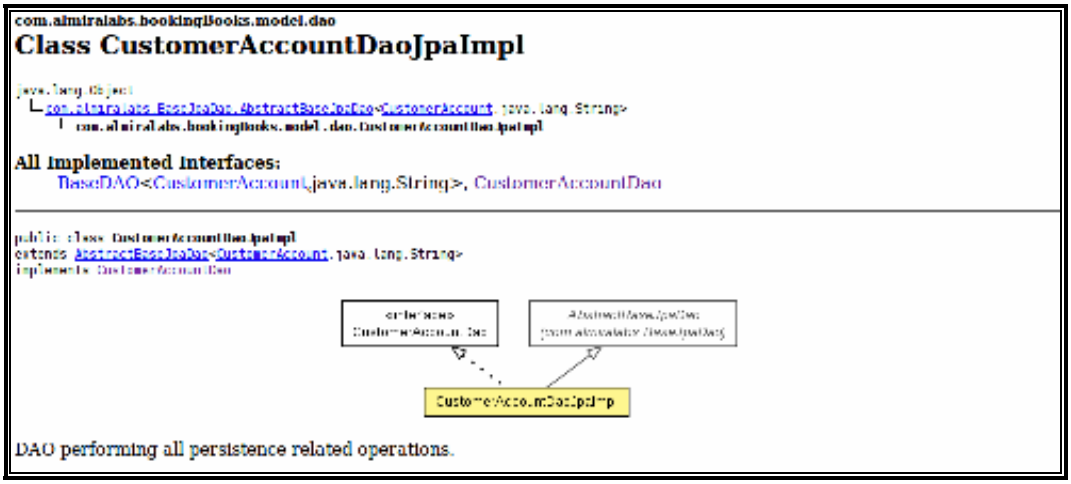
● **Motivation:** This rule allows us to unify criteria when implementing a DAO. In many instances is a DAO interface implementation-specific exceptions (i.e., JDBC). With these exceptions we will add semantic errors that may occur due to an exception.

2.7. Implement the DAO Facades

● **Purpose:** Implement the functionality described in the DAO facades, creating the access code to an object.

● **Motivation:** implement the functionality described in the DAO.

● **Structure:** write the DAO code in methods (services) of the facade.



Picture III-17: CustomerAccountDaoJpaImpl Class Diagram

Method Detail

findAll

Find all entities of the given class.

Specified by:
findAll in interface

Returns:
A list with all the entities of the given class. If no entities are found an empty list is returned

Throws:
- If any runtime error happens

loadByUsername

Return the CustomerAccount, or an exception otherwise

Specified by:
in interface

Parameters:
- the userName to search

Returns:
the CustomerAccount found for the userName

Throws:

checkCustomerAccountEmailAddressAlreadyExist

Method to know if a CustomerAccount email address exist or not

Specified by:
in interface

Parameters:
- the email address to search

Returns:
true or false if CustomerAccount already exist or not

load

Description copied from interface:
Loads a previously created entity.

Specified by:
in interface

Parameters:
- Bussiness key of the entity to be retrieved

Returns:
the found entity instance

Throws:
- If no entity is found with the given id
- If any runtime error happens

Picture III-18: CustomerAccountDaoJpaImpl Method Summary

2.8. Create Entities for Each Business Object

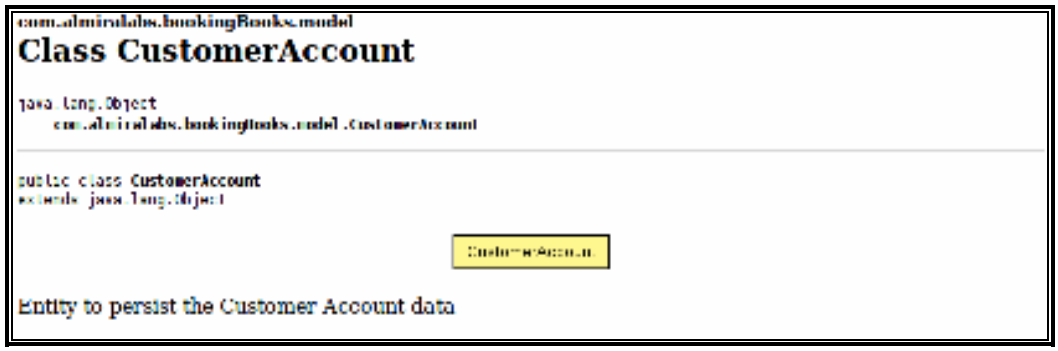
- **Purpose:** Encapsulate an object in a business class that can be manipulated by all sections of the application.

- **Motivation:** The business objects are manipulated (CRUD operations) in many components of an application. The entities encapsulate both the attributes and the rules of a domain object. A business object may consist of a complex object structure, with cardinality in its attributes. For example, a car business object

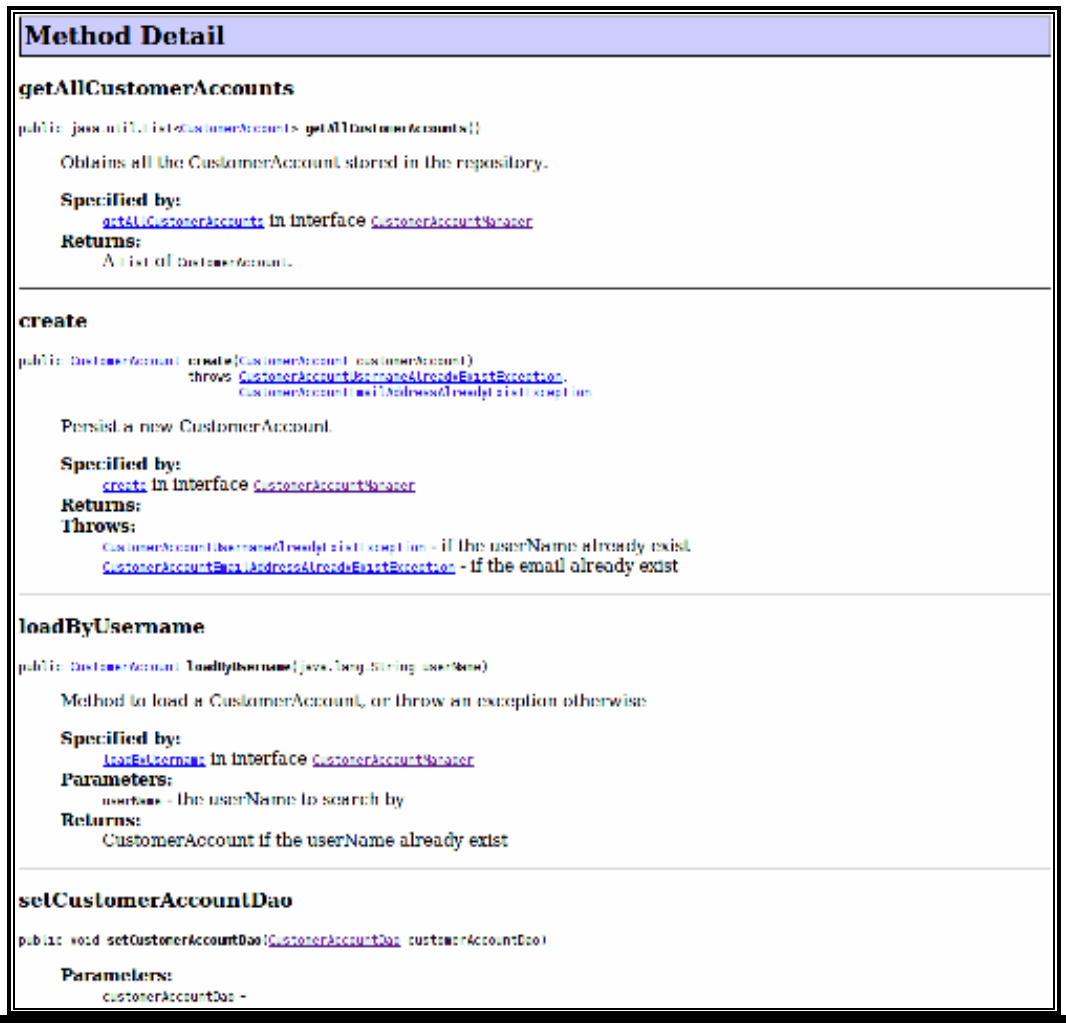
attributes can be as a collection of components, being itself a component class with attributes of part and reference price.

- **Structure:** For each business object defined in the data dictionary, create one or more classes and add the *JPA* annotations.

The class which represents the business object must usually provide typical accessors (getter and setter) for each attribute.



Picture III-19: CustomerAccount Class Diagram



Picture III-20: CustomerAccount Method Summary

2.9. Cover Managers with Technology-Dependent Classes

● **Purpose:** Uncoupling the manager facades of the underlying technology, so they could be reused in different technological contexts.

● **Motivation:** The facades, which are the entry point to the implementation of the functionality of the use case, should not be dependent on a technology. By avoiding this dependence we get a reusable approach.

● **Structure:** The solution used is dependent on the technology selected. Thus, for a standard **web solution**, a solution should implement *Model - View - Controller* (MVC), such as Struts or simply *Servlets + JSP*. For a **web service** technology, we should write a class that would turn the adapter between the *SOAP* message and the facade.

The responsibility of this class is to be an adapter between the outside world and the system we are developing.

I recommend to write a handler for each method of the facade, and return the information expected by the customer. For web Applications, the feedback data are usually *HTML*, so I delegate in *JSP*.

2.10. Dependency Injection Design Pattern

● **Purpose:** To define a mechanism for the composition of dependencies between components.

● **Motivation:** An application consists of many components, each one with clearly identified responsibilities and reusable for any other purpose.

A bad practice to get a component is instantiate it directly, because it can have some construction environment dependencies (name of data sources, configuration items); more over, we bind ourselves to an implementation and not to an interface. Another problem is that it makes us impossible to perform unit testing, since we can not simulate the behaviour of its components.

The dependency injection is a technique for decoupling highly dependent software components, by which each class inject the components you need to carry out their task.

We just need to create a constructor with all its components of the class. This mechanism is called constructor injection. Another valid mechanism is called setter injection, which defines a setter method for each component of the class.

For dependency Injection (also know as Inversion of Control) I will user *Spring*. In the file *spring-config.xml*, is defined the beans dependencies.

Picture III-21: Spring Configuration File

4.2.1.21. Sun Java Code Conventions

1. Description

Sun Java Code Code Conventions, is a document that contains standard conventions that Sun recommends to follow. The rules cover file names, directory organization, indentation, comments, statements, judgments, blanks, parts lists, programming practices, and include code examples.

Code conventions are important to programmers for a number of reasons:

- 80% of the lifetime cost of a piece of software goes to maintenance.
- Hardly any software is maintained for its whole life by the original author.
- Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.
- If you ship your source code as a product, you need to make sure it is as well packaged and clean as any other product you create.

More information can be found in http://java.sun.com/docs/codeconv/CodeConventions.pdf.

4.2.1.22. Bug Life Cycle

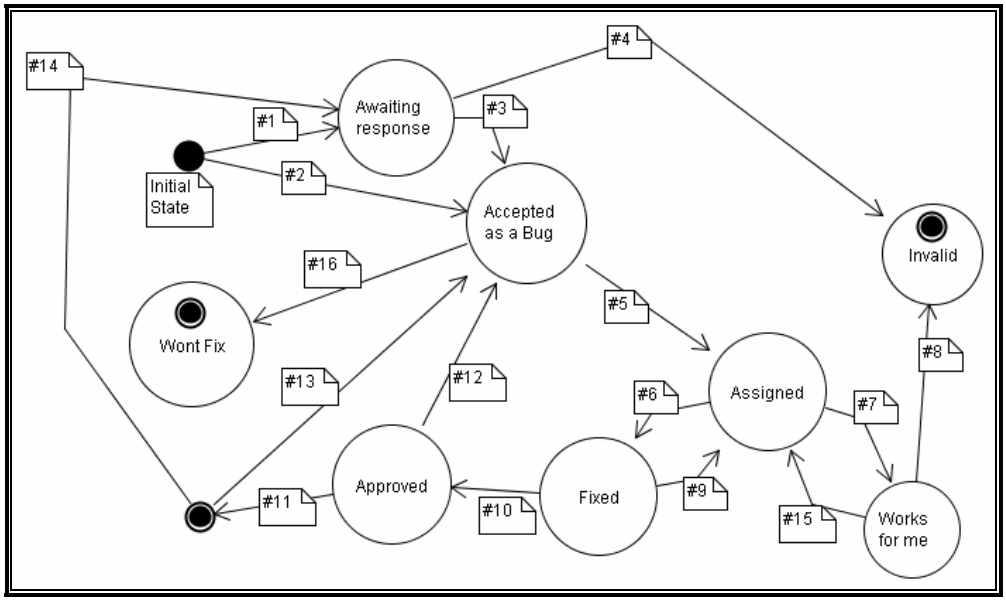
1. General Bug Life Cycle Diagram

Coding bug life cycles are similar to software development life cycles. At any time during the software development life cycle errors can be made during the gathering of requirements, requirements analysis, functional design, internal design, documentation planning, document preparation, coding, unit testing, test planning, integration, testing, maintenance, updates, re-testing and phase-out.

Bug life cycle begins when a programmer, software developer, or architect makes a mistake, creates an unintentional software defect, i.e. bug, and ends when the bug is fixed, and the bug is no longer in existence.

What should be done after a bug is found? When a bug is found, it needs to be communicated and assigned to developers that can fix it. After the problem is resolved, fixes should be re-tested. Additionally, determinations should be made regarding requirements, software, hardware, safety impact, etc., for regression testing to check the fixes didn't create other problems elsewhere.

If a problem-tracking system is in place, it should encapsulate these determinations. A variety of commercial, problem-tracking, management software tools are available (I will use Jira to manage the bugs; some examples will be exposed later). These tools, with the detailed input of software test engineers, will give the team complete information so developers can understand the bug, get an idea of its severity, reproduce it and fix it.



Picture III-22: Bug Life Cycle Diagram

2. General Bug Life Cycle States Transition Diagram

During the life of a bug, it passes through some states. For example, it the *Id 1* of the next table, we can see a new bug reported by somebody different than QA. It must be opened in *Jira*, and assigned to a QA, fill the found in release field in *Jira*, and link to its use case. In the next state (awaiting response) somebody of the QA team will see if it is really a bug or was opened by error, and will be able to accept it as a bug, or link it to the invalid state.

This work flow is specified in the next table.

Id	Initial Resolution	Next Resolution	Initial State	Next State	Reason	Action(s)
1		Awaiting Response.		Open.	Somebody different than QA reports it.	-Assign to QA. -Fill Found in Release field. -Link to affected UC.
2		Accepted as a Bug.		Open	QA tests and discovers it	-Assign to Technical lead. -Fill Found in Release field. -Create task in Test Cycle Definition or link with existing one, remove link to UC, in any case put priority of the Test Case to maximum level (aging policy). -Fill Severity according to Bug Severity definition.
3	Awaiting Response.	Accepted as a Bug.	Open.	Open.	QA tests and checks it in controlled environment.	-Assign to Technical lead. -Create task in Test Cycle Definition or link with existing one, in any case put priority of the Test Case to maximum level (aging policy). -Fill Severity according to Bug Severity definition. Remove link to UC if it exist.
4	Awaiting Response.	Invalid.	Open.	Closed.	QA checks it in controlled environment.	-Assign to reporter.
5	Accepted as a Bug.	Assigned.	Open.	Open.	Technical lead and QA assign it to developer after prioritization session.	-Assign to Developer.
6	Assigned.	Fixed.	Open.	Open.	Developer has fixed it.	-Assign to QA. -Attach patch if in stabilization branch otherwise commit to main line.

						-Fill Fixed in Release check box.
7	Assigned.	Works for me.	Open.	Open.	Developer does not reproduce it.	Assign to QA.
8	Works for me.	Invalid.	Open.	Closed.	QA does not reproduce it either.	Assign to nobody.
9	Fixed.	Assigned.	Open.	Open.	Test case fails.	Assign to Developer (same one).
10	Fixed	Approved	Open.	Open.	QA tests with the patch.	Assign to QA (for re-testing during test-cycles). Commit patch.
11	Approved.	Approved.	Open.	Closed.	QA executes associated task in first Test Cycle execution for this bug.	-Assign to reporter. -If test is not fully automated, lower by one the priority level in Test Cycle Definition (aging policy).
12	Approved.	Accepted as a Bug.	Open.	Open.	Associated task in first Test Cycle execution fails when QA executes it.	-Assign to Technical lead (for prioritization and assignment to developer).
13	Approved.	Accepted as a Bug.	Closed.	Open.	Regression during (non first) Test Cycle execution by QA.	Assign to Technical lead. Put priority of the task in Test Cycle Definition to maximum level (aging policy).
14	Approved.	Awaiting Response.	Closed.	Open.	Somebody different than QA reports it again.	Assign to QA.
15	Works for me.	Assigned.	Open.	Open.	QA re-tests the bug.	Assign to (same) Developer.
16	Accepted as a Bug.	Won't Fix.	Open.	Closed.	Technical lead decides for business reason after prioritization session.	Assign to reporter.

3. Bug Severity Definition

When a bug is created, we must know the bug severity, and fill it in *Jira*. This field is very important, because we can release and deploy a version to a production environment with some trivial bugs; but an only one blocker or critical bug can abort the release process and the deployment of the application, due its importance.

Blocker:	
	-Bugs that make the whole system to shutdown stall or otherwise be completely unusable.
	-Bugs that leave the system in an inconsistent state so that other users or next session of the same user cannot make use of the system.
Critical:	
	-There is a loss for the system's organization of a critical asset, for instance money in a financial transaction.
	-There is a security breach exploitable by the bug.
Major:	
	-Bugs that leave a use case completely unusable (an HTTP 500 or 404 or blank page very soon in the flow).
Normal	
	-Bugs that cause the basic flow or any alternative flow to completely fail.
	-In some step of the given flow the user gets a HTTP 404 or 500 or blank page stopping abruptly the expected sequence of steps of the flow. Beware if such type of errors appears very soon in the flow leaving the use case completely unusable that would be a Major bug.
Minor	
	-Something not working as specificied but either there is a workaround for the user or the flow can be finished without state inconsistencies or 404, 500 HTTP errors or blank pages.
Trivial:	
	-Types and wrong literals.
	-Look & feel issues: Buttons that should not appear, Default values for fields, Confirmation popups, Wrong redirections, Colors, Wrong html styles.

4.2.1.23. USDP Software Life Cycle

1. Description

The Unified Software Development Process (USDP) is a set of activities needed to transform a user's requirement into a software system. The USDP, is component-bases, which means that the software system is made up of software components intercommunicate via well-defined interfaces.

2. Features

The aspects of the unified process are the next: use case driven, architecture-centric, and iterative and incremental.

- **USDP is Use-Case Driven**

A use case is a piece of functionality in the system that gives a user a result of an action. A use case captures functional requirements. While is true that a use case drives the process, they are not selected in isolation. They are developed in tandem with the system architecture. That is, the use case drives the system architecture and the system architecture influences the use cases. Therefore, both the system architecture and the use cases mature as the life cycle continues.

- **USDP is Architecture-Centric**

The role of software architect is similar in nature to the role architecture plays in building construction. The building is looked at from various viewpoints: structure, heat construction, plumbing, and so on. This allows a builder to see a complete picture before construction begins. Similarly, architecture in a software system is described as different views of the system being built.

The software architect concept embodies the most significant static and dynamic aspects of the system. The architecture grows out of the needs of the enterprise, as sensed by users and other stakeholders, and as reflected in the use cases. However, it is also influenced by many other factors such as the platform the software is to run on, the frameworks used, deployment considerations, legacy systems, and non-functional requirements. Architecture is a view of the whole design with the important characteristics made more visibly by leaving details aside.

The architecture must be designed so as to allow system to evolve, not only through its initial development but through future generations.

- **USDP is Iterative and Incremental**

Develop a commercial software product is a large undertaking that may continue over several months to possibly a year more. It’s practical to divide the work into smaller slices or mini-projects- Each mini project is an iteration that results in an increment. Iteration refers to steps in the workflow, and increments, to growth in the product. To be more effective, the iteration must be controlled; that is they must be selected and carried out in a planned way. This is why they are mini-projects.

In every iteration, developers identify and specify the relevant use cases, create a design using the chosen architecture as a guide, implement the design in components, and verify that the components satisfy the use cases. If an iteration meets its goals, developers proceed with the next iteration. When an iteration does

not meet its goals, the developers must revisit their previous decision and try a new approach.

3. Benefits of a Controlled Iterative Process

• Controlled iteration reduces the cost to the expenditures on a single increment. If the developers need to repeat the iteration, the organization loses only the misdirected effort of one iteration, not the value of the entire product.

• Controlled iteration reduces the risk of not getting the product to market on the planned schedule. By identifying risk early in the development, the time spend resolving them occurs early in the schedule when people are less rushed than they are late in the schedule

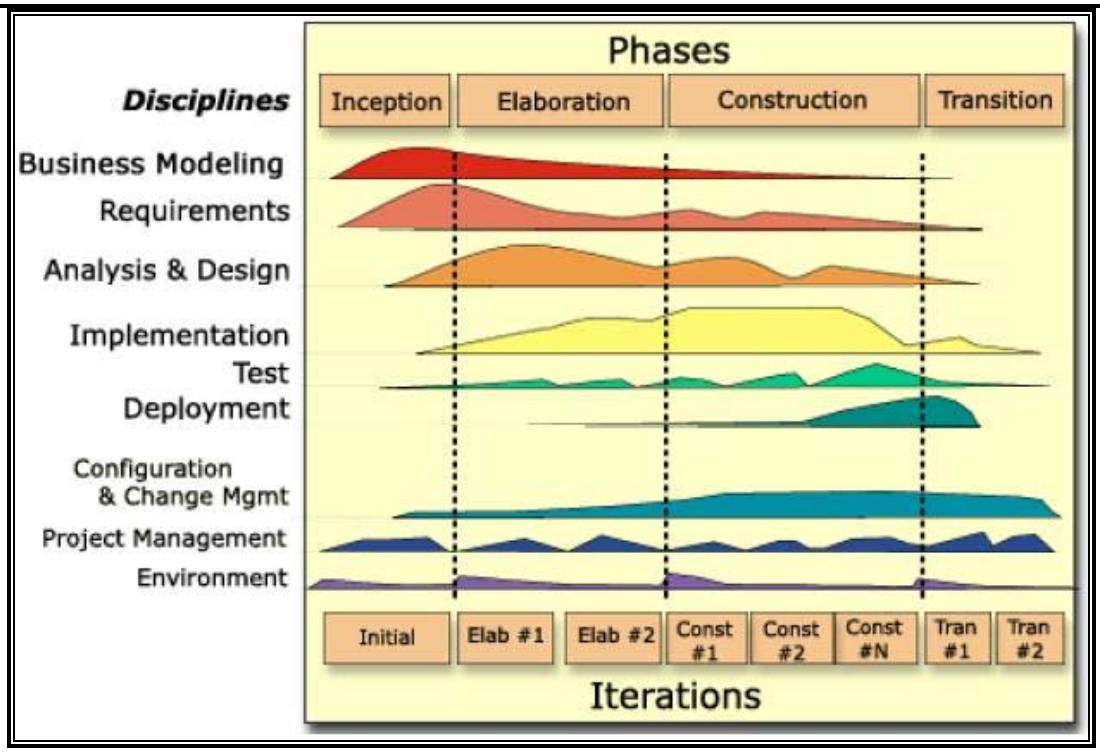
• Controlled iteration speeds up the time of the whole development effort because developers work more efficiently toward results in a clear, short focus period of time, than in the long schedule.

• Controlled iteration acknowledges a reality often ignored – that users needs and the corresponding requirements cannot be fully defined up front. They are typically refined in successive iterations. This mode of operation makes it easier to adapt to changing requirements.

These concepts: use-case driven, architecture-centric and iterative and incremental development are equally important. Architecture provides the structure in which to guide the work in the iterations, whereas use cases defined the goals and drives the work of each iteration.

4. Phases within a Cycle

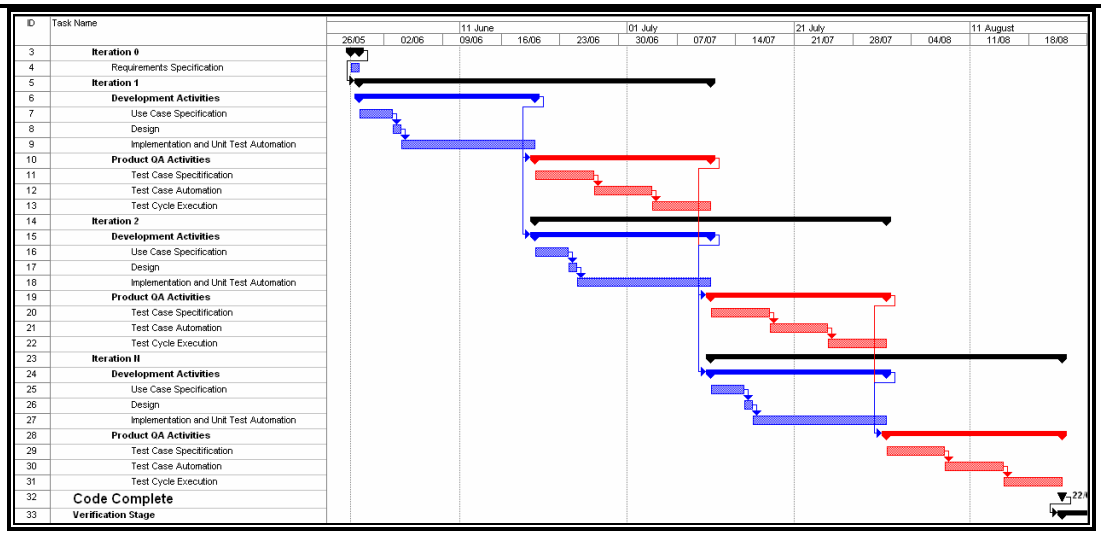
Each cycle takes place over time. This is, in turn, is divided in four phases (Inception, Elaboration, Construction and Transition).



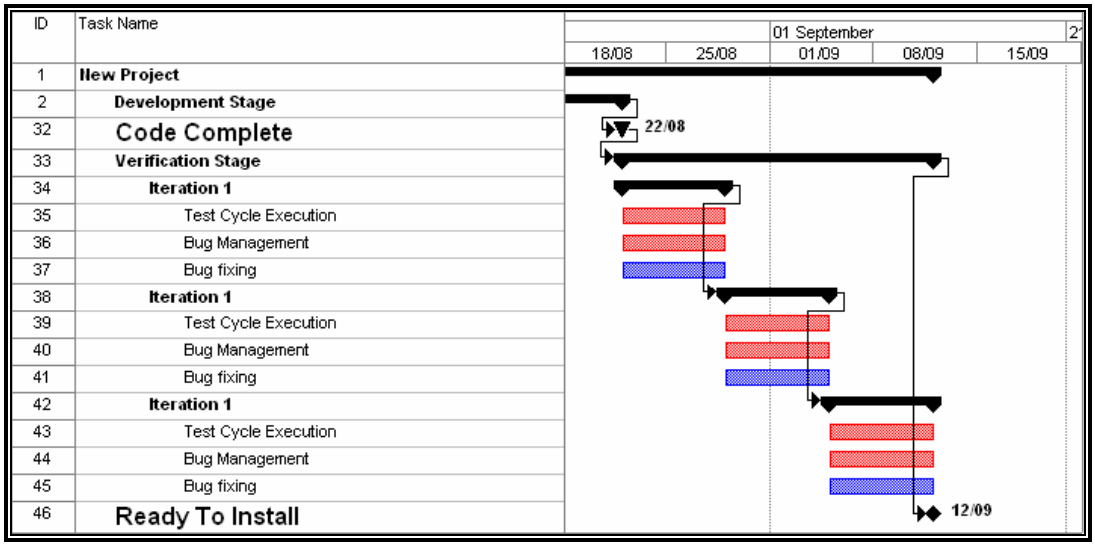
Picture III-23: USDP Development Phases

5. Development Stage Fases

Once we have a c*ode complete*, we are ready to move to the next stage. *Test cases execution* is the only activity allowed in this phase (and *test cases specification/automation* in some cases). This means that new functionality is never added during this phase. Tests are performed against a frozen product version. This phase can also contain several iterations (*test cycles*). Validation involves a group of tasks performed at different levels and moments. Bear in mind that you cannot fix a bug in the current version of the product since it has already been closed. Also consider that there are two teams working with different purposes. Quality assurance members run the tests and report about the state of bugs whereas development members are in charge of fixing them.



Picture III-24: Gantt Development General Iterations Diagram



Picture III-25: Gantt Development Diagram for a Single Iteration

4.2.1.24. Quality Policy Example

1. <u>Description</u>

The quality policy is the top management's expression of its intentions, direction, and aims regarding quality of its products and processes.

2. <u>Booking Books Quality Policy Example</u>

Booking Books and its affiliated corporate entities, is committed to the total customer satisfaction through quality in product development, support and service.

Booking Book's management is committed to build on its industry leadership by providing total quality in customer

satisfaction and in vendor performance. Our mission is to supply our customers with on-time delivery of products, which are of the best possible quality and consistently meet their design specifications and performance criteria.

In order to live up to our quality policy and objectives, **Booking Books** shall supply software development products efficiently, economically and on time with standards that consistently meet or exceed our customers' requirements. This is accomplished through a continuing program to improve upon operational procedures and systems while striving to supply our customer's products with zero defects.

Products and service provided by **Booking Books** shall conform to the requirements specified by our customers, and be manufactured under *ISO 9001* Quality Standards. Management's responsibility is to provide direct leadership and resources to ensure continued conformance with these requirements. A cornerstone of **Booking Book's** quality policy is that excellence in product quality and customer service is the result of collective effort and commitment from all its team members. This commitment to quality shall be realized from product design to after-sales support.

Objective Evidence in support of **Booking Book's** quality policy is the responsibility of the Vice President, Corporate Quality Control.

4.2.1.25. Quality Plan Document Template

1. <u>Description</u>

In this paragraph is explained a software quality plan. The purpose of this Software Quality Plan is to establish the goals, processes, and responsibilities required to implement effective quality assurance functions.

It provides the framework necessary to ensure a consistent approach to software quality assurance throughout the project life cycle. It defines the approach that will be used by the Software Quality personal to monitor and assess software development processes and products to provide objective insight into the maturity and quality of the software.

2. <u>Scope</u>

This plan covers SQA activities throughout the software life cycle phases.

3. <u>Management</u>

This section describes the management organizational structure, its roles and responsibilities, and the software quality tasks to be performed.

- **3.1. Management Organization**

Project efforts are supported by numerous entities, organizations and personnel. Relevant entities/roles that are of interest and applicable to this SQA Plan and the software assurance effort are described at a high level below.

- **3.2. Tasks**

This section summarizes the tasks (product and process assessments) to be performed during the development, operations, and maintenance of software. These tasks are selected based on the developer's Project Schedule, Software Management Plan (SMP) and planned deliverables, contractual deliverables, and identified reviews. Reference Procedure for Developing and Implementing Software Quality Programs, for additional information on the various process and product assessments. Reference Software Quality Assessment Process, for specific instructions on conducting process and product assessments.

- **Product Assessments**

The following are typical product assessments that may be conducted by SQ personnel. See the SQ Activity Schedule for the planned assessments:

- Peer Review packages

- Document Reviews

- Software Development Folders

- Software Configuration Management (e.g., configuration baselines, configuration change requests, and change control records)

- Test results (e.g., requirements traceability matrix, test reports).

- **Process Assessments**

The following are typical process assessments that may be conducted by SQ personnel. See the SQ Activity Schedule for the planned assessments:

- Project Planning

- Project Monitoring and Control

- Measurement and Analysis

- System/Subsystem Reviews

- Peer Reviews

- Requirements Management

- Software Configuration Management and Configuration Audits (FCA/PCA)

- Test Management (Verification & Validation)

- Software Problem Reporting and Corrective Action

- Risk Management

- Supplier Agreement Management

● 3.3. Roles and Responsibility

This section describes the roles and responsibilities for each assurance person assigned to the Booking Books project.

● 3.4. Software Assurance Estimated Resources

Staffing to support software assurance (i.e., quality, safety, and reliability) activities must be balanced against various project characteristics and constraints, including cost, schedule, maturity level of the providers, criticality of the software being developed, return on investment, perceived risk, etc.

4. Documentation

This section identifies the minimum documentation governing the requirements, development, verification, validation, and maintenance of software that falls within the scope of this software quality plan. Each document below shall be assessed (reviewed) by SQ personnel.

● Minimum documentation requirement

- Quality Manual

- Software Assurance Plan

- Software Management Plan

- Configuration Management Plan

- Software Requirements Specification

- Risk Management Plan

- Software Safety Plan

- Test Plans (Verification and Validation)

- Software User’s Guide

- Software Maintenance Plan

- Interface Control Document(s)

- Test Reports and Artifacts

- Software Version Description Document (VDD)

- Software Requirements Traceability Matrix

- Software Development Records

- Peer Review data packages

5. Standards, Practices, Conventions, and Metrics

This section highlights the standards, practices, quality requirements, and metrics to be applied to ensure a successful software quality program.

Personnel are governed by software quality procedures, work instructions, checklists, and forms developed and approved by the company. These practices and conventions are tools used to ensure a consistent and objective approach to software quality for all projects. SQ personnel are also experienced in the Software Engineering Institute Capability Maturity Model Integration (SEI-CMMI) methodology and are applying generic and specific practices for Process and Product Quality Assurance.

6. Software Reviews

This section identifies the number and type of system/subsystem reviews and engineering peer reviews that will be supported by the SQ Personnel. The Software Management Plan (SMP), the project milestone chart, the project's Engineering Peer Review Plan, and the SQ Personnel resource levels determine the reviews that are supported.

- **Minimum Software reviews**

For each review, SQ will assess the review products to assure that review packages are being developed according to the specified criteria, the review content is complete, accurate, and of sufficient detail, and Requests for Action are captured, reviewed, and tracked to closure. In addition, SQ will assess the processes used to conduct the reviews to determine if appropriate personnel are in attendance, correct information is presented, entry and exit criteria are met, and appropriate documents are identified for update.

The following software reviews may be assessed by SQ:

- System Concept Review

- Software Specification Review

- Preliminary Design Review

- Critical Design Review

- Test Readiness Review

- Acceptance Review

- Operational Readiness Review

- Mission Operations Review

- Flight Operations Review

- Peer Reviews

7. Tests

SQ personnel will assure that the test management processes and products are being implemented per the Software Management Plan and /or Test Plan(s). This includes all types of testing of software system components as described in the test plan, specifically during integration testing (verification) and acceptance testing (validation).

SQ personnel will monitor testing efforts to assure that test schedules are adhered to and maintained to reflect an accurate progression of the testing activities. SQ will assure that tests are conducted using approved test procedures and appropriate test tools, and that test anomalies are identified, documented, addressed, and tracked to closure. In addition, SQ will assure that assumptions, constraints, and test results are accurately recorded to substantiate the requirements verification/validation status.

SQ personnel will review post-test execution related artifacts including test reports, test results, problem reports, updated requirements and so on.

8. Tools, Techniques and Methodologies

7.1. Development tools

-Eclipse

…

7.2. Software quality tools

-Jmeter

…

7.3. Project Tools

-Jira

…

9. Record Collection, Maintenance and Retention

SQ personnel will maintain records that document assessments performed on the project. Maintaining these records will provide objective evidence and traceability of assessments

performed throughout the project's life cycle. Example records include the process and product assessments reports, completed checklists, the SQ Activity Schedule, metrics, weekly status reports, etc. For more details on SQ records, their location, and data retention, reference the OSSMA Software Quality Assurance Data Management Plan.

10. Training

SQ personnel shall have fundamental knowledge in the following areas/disciplines through prior experience, training, or certification in methodologies, processes, and standards:

- Software Quality Assurance:
- Audits and Reviews
- Risk Management
- Configuration Management
- Software Safety
- Contracts/Contractor Surveillance
- CMMI
- ISO 9001
- Project-specific Training

It is the responsibility of the SQ personnel to acquire the necessary skills or knowledge in each of the above disciplines. An SQ Training log has been prepared that specifies the type of training and/or on-the-job experience that has been completed, along with the source of the training, and the date of completion.

11. Risks Management

SQ personnel will assess the project's risk management process against the <Project Name> Risk Management Plan and GPG 7120.4. SQ participates in <weekly/monthly> risk management meetings and reports any software risks to the SAM and the project's Risk Manager.

12. SQA Plan Change Procedure and History

SQ personnel are responsible for the maintenance of this plan. It is expected that this plan will be updated throughout the life cycle to reflect any changes in support levels and SQ activities. Proposed changes shall be submitted to the Booking Books Systems Assurance Manager, along with supportive material justifying the proposed change. Changes to this document require prior approval of the Booking Books Project.

4.2.1.26. Risk Management Document Template

1. Description

All projects contain elements of risk. Risk refers to a positive or negative uncertainty. The possibility that a given task is completed ahead of schedule is an example of positive risk. The possibility that a vendor will not deliver the hardware on time is an example of negative risk.

The risk management process contains the following elements.

1. Risk identification: Define the risk issue by root cause

2. Risk qualification: The high, medium, or low impact it can have on the project

3. Risk quantification: The probability, schedule, and cost impact

4. Risk response: Strategy and activities to manage the risk

5. Risk control: How the responses will be monitored and controlled

As a number of risk issues are identified, qualified, and quantified should be dome the risk response.

The project manager begins by identifying the risk elements in the project plan.

Once the risk elements have been identified, the project manager determines how to avoid, mitigate, transfer, or accept the risk.

-**Risk Avoidance**: Take immediate actions such as adding activities to the program schedule or provide direction to a vendor to preclude the risk event from occurring.

-**Risk Mitigation**: Adjust the Program Plan to reduce the potential impact of the risk event. Typical examples include time buffers and risk funding.

-**Risk Transfer**: In some cases the optimal course of action is deflecting the risk issue to a third party. In large complex programs this would, for example, take the form of requiring Contractors to secure performance bonds.

-**Risk Acceptance**: There are times when the reality of a risk event must be accepted and simply planned for. In those cases, the program manager creates a set of backup plans that will go into immediate effect when and if the risk event materializes. This might include, for example, a backup schedule that is ready if and when a Sole Source Supplier's materials (though ordered early) are nevertheless delivered late.

2. Risks Template

Nº	Risk Event Description	Qualification H=High M=Medium L=Low	Impact (1-10) / Qualification (Euros value)	Status	Planned Response (Avoid, Mitigate, Deflection, Retain)

4.2.1.27. Recruitment Interview Template

1. Description

I have done an example of a recruitment interview template, for the human resources recruitment process.

2. Recruitment Interview Template

2.1. General Software Process

- **Describe the process or processes you know and have used.**

The candidate should describe real examples as much as possible, and if he mentions any iterative process and use case based process should explain how he planned iterations and chose use cases to concentrate on. Desirably he should mention peer-reviews and code inspections, some testing procedures or configuration management.

- **What type of process is that one you just described?**

The candidate should clearly respond one of the following: Waterfall, Iterative, Incremental, Iterative-incremental or Spiral.

- **If you had to drive an iterative-incremental development, what would you use as a criteria to divide the work in iterations?**

The candidate should use some kind of functional division of the system taking into account dependencies among them.

2.2. Requirements

- **Define requirements for this user story:** I am a customer and I need you to develop a system with a web interface to final mobile subscribers. The functionality in which I want you to concentrate is the one about transferring part of the subscriber's

pre-paid balance to another subscriber. Suppose you know how to interconnect the system to the IN network and obviate the details. Please concentrate in asking me the relevant information and describing the final functionality as it should passed to your technical team and to me, as customer, for approval.

- Functional requirements: subscribers should be able and need to securely log in the system providing mobile number and password; system should provide a mechanism via web to ask for a new password to be sent to the terminal through SMS; system should allow a logged subscriber to view his current balance; system should allow to transfer part of his/her balance to another mobile number.

- Non functional requirements: interface with given IN system; performance: (anything here); legal (anything);

● Should somebody outside the involved team review requirements?

Yes, because document should not be ambiguous, in order to make sure they understandable by anybody not involved in the team and that these requirements are traceable and testable automatically.

2.3. Use Cases

● Can you tell what structure a use case should have?

Triggers, pre-conditions, main flow, alternative flows and post-conditions.

● What is the traceability between requirements and use cases?

The traceability is the relationship between requirements and use cases, and vice versa, which defines which requirements are covered by which use cases, and which use cases can be used to verify which requirements.

2.4. Design

● What things would you search for in a code inspection?

Among the things the candidate might mention should be: code standards, code metrics, obvious design flaws, like not using patterns or bad OO practice, code not being in sync with the design, algorithmic complexity, like using exponential (recursive) or logarithmic solutions when there exists an obvious polynomial solution, concurrency problems, bad use of threads and synchronization: dead locks, live locks and starvation.

● What is an obvious sign of bad code in Java or any other OO language?

The candidate could answer any of the following: methods too long (more than a screen full); low cohesion: the class does a lot of things not related to each other, it has too many responsibilities, strong coupling: a class knows things about the inner variables or working of another class, lack of appropriate comments in code.

2.5. Code Review

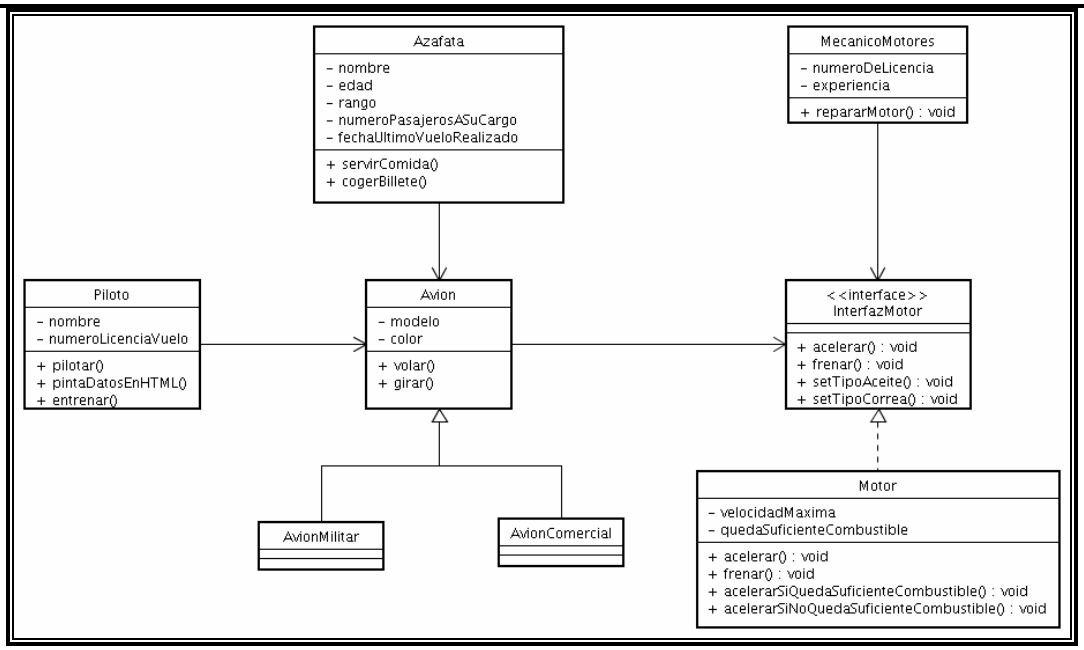
● Review the following class and tell us what is wrong, and how you would refactor it?

```
01. class Sender {
02.     public void send(String type, String message, String to, String subject) {
03.         if (type == "SMS") {
04.             SMPP smpp = new SMPP("192.168.1.4", "4000");
05.             smpp.openConnection("your username", "your password");
06.             smpp.send(to, message, null, "1h");
07.             smpp.closeConnection();
08.         } else if(type == "mail") {
09.             SendMailConnection smc = new SendMailConnection("mail.myserver.com");
10.             smc.prepareMessage(to, message, subject);
11.             smc.send();
12.             smc.close();
13.         } else {
14.             throw new RuntimeException("Type is unknown");
15.         }
16.     }
17. }
```

Picture III-26: Sender Recruitment Interview Class

Use factory instead of if (type), introduce logs, introduce exception handling, remove connection and close from send method (e.g. connection pool), introduce Java docs, do not use Runtime Exception, hard-coded constants should be in configuration files.

● Review the following class diagram and suppose the semantic of the diagram is right. Could you tell us what is wrong and how you would redesign it?



Picture III-27: Recruitment Interview Class Diagram

- *Azafata* class is loosely coupled. *Nombre*, *edad* and *rango* attributes tells something about the *Azafata* person, but *numeroPasajerosASuCargo* and *fechaUltimoVueloRealizado* tells something about *Vuelo*, that is the relation between *Azafata* and *Avion*.

- *Avion* methods should be extracted to an interface and maybe an abstract class. A factory class should do the creation of concrete *Avion* types.

- The method *pintarDatosEnHTML()* of *Piloto* class should be extracted to another class who knows how to do it. It is not acceptable to mix business logic with presentation logic.

- *InterfazMotor* should be split in two interfaces. One should include the operations the *Piloto* uses (accelerate and break) and the other should include the operations used by the *MecanicoMotores* (*setTipoAceite* and *setTipoCorrea*).

- A state pattern should be included in the Motor class. *Acelerar* operation depends on the internal state of the class (concretely on the *quedaSuficienteCombustible* attribute) to be performed.

2.6. Testing

● **How do you ensure that what you deliver is what you were asked for?** By checking that requirements are fulfilled by the deliverable through tests.

- What do you use to define those tests? Use cases.

● **When should the testing process (not just the application testing) start?** As soon as possible, at least the definition of the functional test cases taken out of the use cases or functional requirements, and the coding of the functional tests should start along with the rest of the code.

2.7. Procedures

● **What kind of tests should be done?**

Unit tests, for each class there should be a test for the public interface. Integration tests, the same but only for the main API or system interface to be used by the final application in an integrated environment as similar as possible to the production one.

● **What is the difference between black-box and white-box tests?**

Black-box tests do not know any implementation detail and white-box tests know how the tested functionality has been implemented.

● **How would you do a unit test case if your class is calling to another one and that one is not yet available?**

I would code *Mock-Up* classes or *stubs* which have the same interface and emulate a pre-fixed valid response.

● **With which tools or frameworks have you worked for testing?** *JUnit*, *Cactus*, *WebLoad*, *HtmlUnit*, *Mercury*, *HttpUnit*, test cases generators, and so on.

2.8. Metrics

● Do you know what code metrics are? Can you name some simple metric?

Metrics are measures in its widest sense but they can be used for two purposes, firstly for project tracking, control and future estimations and by the other hand for quality control. Code metrics are measures about the code, which can be used to get an idea of its quality. Some simple metrics are lines of code (LOC), cyclomatic complexity, comments per line of code, depth of inheritance tree, FAN-IN (number of modules that give information to a module), FAN-OUT (number of modules that get information of a module), etc.

● **Do you know what cyclomatic complexity is?**

Will a simple method have a high cyclomatic complexity? Why? It is a code metric that measures the complexity of the code. A simple method would have a low cyclomatic complexity, because this metric grows as the method has more control flows.

● What will you think if a project has an inheritance tree with a medium depth of seven?

The classes of the project have too much inheritance. So the project will not be easily understandable, and it could be difficult to test.

2.9. Estimation

● What formal academic techniques do you know for estimation?

The answer may vary, from complex methods (like *COCOMO*), to techniques (like f*unction points*). It would be very interesting that the candidate knows something about f*unction points* or the newer *use case points*.

2.10. Software Configuration Management

● Can you define what SCM (Software Configuration Management) is?

A discipline applying technical and administrative direction and surveillance to: identify and document the functional and physical characteristics of a configuration item, control changes to those characteristics, record and report change processing and implementation status, and verify compliance with specified requirements.

● Can you tell us the difference between a branch and a tag?

A tag is a photo of a configuration in a certain moment. A branch is a new configuration that evolves through time independently from trunk or other branches.

● Can you define what a baseline is?

A baseline is the compilation of all documentation (plus source code files) that represents a software product at a specific point in time.

2.11. Database Modelling

● Explain how you model a class hierarchy in an entity-relation diagram. For example: the Customer and Administrator (each one with its own attributes) classes inherit from the User class (which also has its own attributes). Candidate should describe the three basic inheritance mapping strategies:

- table per class hierarchy: Exactly one table is required. There is one big limitation of this mapping strategy: columns declared by the subclasses may not have NOT NULL constraints.

- table per subclass: Four tables are required. The three subclass tables have primary key associations to the super class table (so the relational model is actually a one-to-one association).

- table per concrete class: Three tables are involved for the subclasses. Each table defines columns for all properties of the class, including inherited properties.

2.12. Human Competencies and Non-Technical Questions

• Did you do any teamwork?

• How do you feel better, working cooperatively in a team or doing most of the work by yourself?

• Why did you choose this company?

• What do you think it could be the biggest problem that you would encounter in the adaptation period?

• What differences do you think that a company like this have with bigger companies?

• What is your availability to travel?

• What is your level of English?

• Almost all of our documentation is in English, do you think you can adapt to read and, more important, to write documents in English?

• Would you mind if we continue this interview for a few minutes in English? (at this time, try to make two or three concrete but very simple technical questions, and make also a few questions non-technical about hobbies, travel, sports, the last book the candidate read, blogs or web pages the candidate reads...).

• What is your current salary?

• What is your availability? In which date could you start to work for our company?

• If the selection process is successful, what will be your level of engagement? Will you willingly accept the signature of a compromise letter or a contract?

• How much time do you think it will take to develop your professional career in this company?

• How do you rate the security at work, speaking in long term?

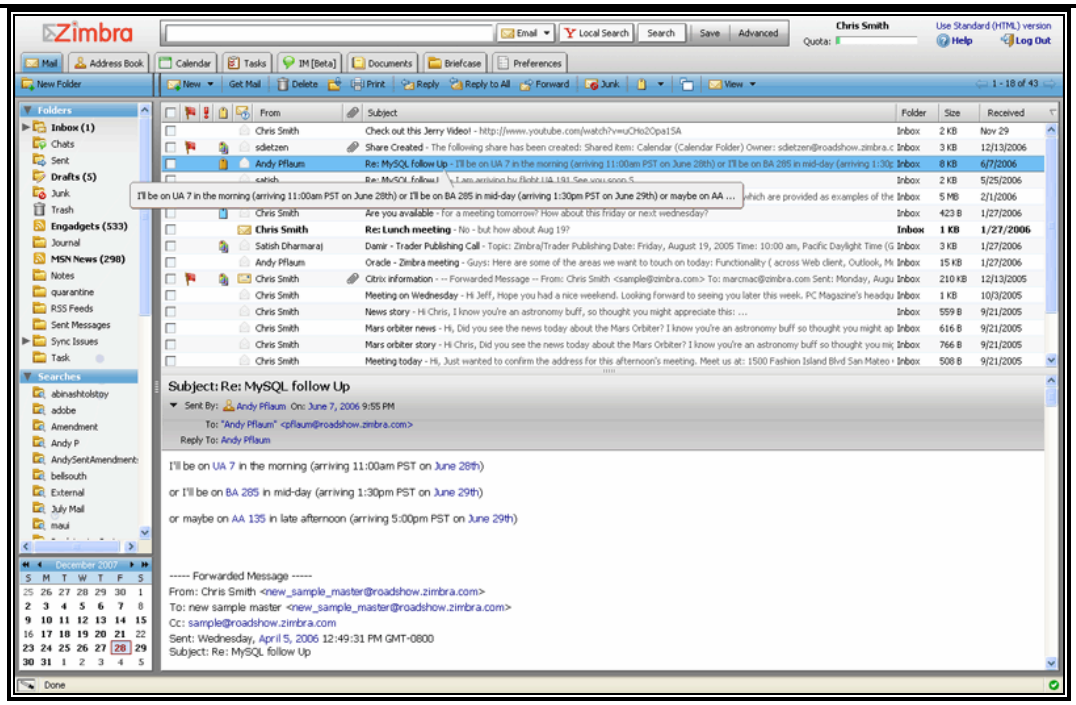
2.13. Human Resource Score Diagram

Nominal	Score	Description
None.	0	The interviewed has no knowledge Theoretical.
Theoretical.	1	It rings a bell to the candidate or it remembers something from University or another source.
Basic.	2	He/she knows it (may be not in a very deep formal way) and is capable of explaining it but it does not have a solid experience using it.
Average.	3	He/she has its own way to applying that knowledge with sufficient experience although not in the best possible (company's or better).
Company's Way .	4	The interviewed is aligned with company's knowledge in the topic.
Advanced.	5	The interviewed has a knowledge above average regarding that held in company or provides with knowledge that is a good value for the company.

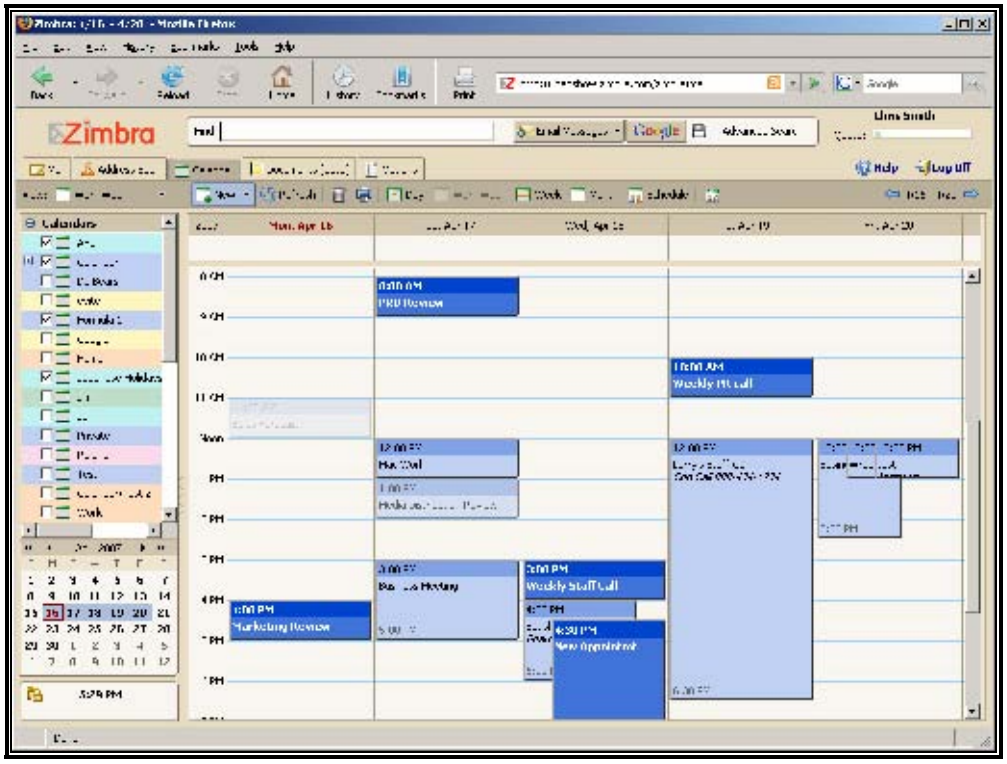
4.2.1.28. Zimbra Collaboration Suite

1. Description

The *Zimbra Collaboration Suite* is a full-featured collaboration suite that supports email, group calendars and document sharing using an Ajax web interface that enables tool tips, drag-and-drop items, and right-click menus in the UI. Also included are advanced searching capabilities and date relations, online document authoring, and a full administration UI.



Picture III-28: Zimbra Email Snapshot



Picture III-29: Zimbra Calendar Snapshot

I will use *Zimbra* as email and calendar server. More information can be found in: http://www.zimbra.com/.

4.2.1.29. SME Authorities Description

1. Description

For a SME, should exist, at least, the next authorities (NOTE: more than one authority can be done by the same person, for example *Project Leaded* and *Developer*):

Corporate Title	Description	Responsible
CEO.	Chief executive is the highest-ranking corporate officer (executive) or administrator in charge of total management of an organization.	-Coordinates external initiatives at a high level. -Establish policies, objectives, goals, and action lines in the activities of the company.
CFO.	The chief financial officer is a corporate officer primarily responsible for managing the financial risks of the corporation. This officer is also responsible for financial planning and record-keeping, as well as financial reporting to higher management.	-Strengthening budget execution and statistics of the company, both qualitatively and quantitatively. -Develop and / or reschedule the budget of the company, according to the pertinent directives. -Prepare and report information of programs and budgets to the government agencies, according to the standards set.
CMO.	Chief Marketing Officer is a corporate title referring to an executive responsible for various marketing in an organization.	-Primary or shared responsibility for areas such as sales management, product development, distribution management, public relations, marketing communications, pricing, and customer service, CMOs are faced with a diverse range of specialized disciplines in which they are required to be knowledgeable. -Recommend prioritization of the implementation of investments and programs.
CTO.	Chief Technical Officer is an executive-level position in a company or other entity whose occupant is focused on scientific and technological issues within an organization.	-Overseeing Research and Development activities, and formulating long-term visions and strategies at the officer level. -Prepare technical studies and reports related to an area. -Propose research and development to ensure compliance with company objectives. -Review and update technical documents necessary for the management of the company as well as driving the recording and dissemination of those.
CQO.	The Chief Quality Officer is responsible for planning, administration, and monitoring of consistent readiness of all quality management, regulatory requirements, and quality improvement processes. The Chief Quality Officer will oversee and coordinate all company	-Ensures and develop a system policy, quality policy and quality objectives for organizational structure and collaborates in the approval of proposed structures. - Ensure that quality processes takes place and are used regarding its effectiveness. -Develops long-range goals, annual objectives, and strategies for area(s) of responsibility for the

	efforts to monitor and maintain compliance with all regulatory, and State. The position reports directly to the CEO.	Quality area, as ISO or CMMi certifications. -Measures and reviews system performance. Develops and, as appropriate, seeks approval of system budget. -Reviews budget performance. -Provides input into major capitol expenses related to quality (e.g., information systems). -Monitors activities for and ensures compliance with laws, government regulations, legal requirements, and system/facility policies. As directed, implements external and internal audit recommendations.
Project Leader/QA Leader.	Project Leader and QA Leader are responsible for facilitating the project, managing the project, and monitoring progress in development and quality areas.	-Track the implementation of development projects. -Delegate tasks, assess risks, and insures the project is remaining within the allotted budget -Continuously coordinate with users, so that their requirements are met. -Make changes and / or additions to systems based on user requirements.
Developer.	Developer is a person or organization concerned with facets of the software development process. They can be involved in aspects wider than design and coding, a somewhat broader scope of computer programming or a specialty of project managing including some aspects of software product management.	-Participation in software product definition, including Business case or Gap analysis. -Create specifications and requirements analysis. -Development and refinement of throw-away simulations or prototypes to confirm requirements. -Feasibility and Cost-benefit analysis, including the choice of application architecture and framework, leading to the budget and schedule for the project. -Design, Implementation (installation, configuration, programming/customization, integration, data migration) and maintain software programs. -Authoring of documentation needed by users and implementation partners and so on. -Testing, including defining/supporting acceptance testing and gathering feedback from pre-release testers. -Participation in software release and post-release activities, including support for product launch evangelism (e.g. developing demonstrations and/or samples) and competitive analysis for subsequent product build/release cycles.
Technical Support.	Technical support services attempt to help the user solve specific problems with a product, rather than providing training, customization, or other support services.	-Maintain company intranet, extranet, backups. -Install and configure equipment (hardware and software). -Train users in the use of commercial software

		itself.

4.2.1.30. Purchase Management Template

1. Description

In the next table I describe the purchase management template, in order to describe the section *7.4.1. Purchasing Process*.

Step	Stage	Description	Responsible
1	Plan.	Selection of suppliers and contractors: This activity can be done via an audit, to those providers selected by the directors of area considered critical suppliers or contractors or that affect the process performance and service.	Directors.
1.1	Plan.	Prepare the annual purchasing budget.	Directors.
2	Perform.	Purchase.	
2.1	Perform.	Detection of the need to purchase: each area generates its monthly purchase requirements under the ordinary course of business, and it sets the annual purchasing budget.	Directors.
2.2	Perform.	Contacted the suppliers and contractors selected depending on the type of goods or services procured.	Each stakeholder shopping.
2.3	Perform.	Preparation of requisition: requisitions were made according to the needs identified and budgeted. The Request for Procurement and / or contractor is delivered to the Purchasing Assistant, who assigns the cost account and review the application of the relevant requirements, including: - Signature of Applicant. - Allocation of Account. - Specific description of the product or service requested. - Annex table comparing quotes and proposals.	Directors.
2.4	Perform.	Approval of Purchases. Develop the contract as applicable. Also requested once the purchase is approved, a compliance policy as collateral and support according to requirements.	Directors.
2.5	Perform.	Follow to Purchase: It is up to the delivery terms, guarantees, quality.., Agreed with the supplier	Directors.
2.6	Perform.	Verification of purchased product or service: validate the characteristics of the product purchased, for compliance with the requirements as specified in the Purchase Order / Service / Contract. If any inconsistency is the return or claim to the provider and request a change or improvement of the service.	Directors.
2.7	Perform.	Inventory Control: create an entry, with a preliminary inventory listing prepared. An annual physical inventory performed can be	Directors.

		done by each Director.	
3	Verify.	Follow the supplier according to the product performance in relation to the purchased product.	Directors.
4	Improve ment.	Meetings are held to improve key aspects of supplier performance.	Directors.

4.2.1.31. Software Customer Satisfaction Survey

1. Description

Booking Books is committed to quality and customer satisfaction. We would like to know your opinion of our *Booking Books* software product. Please indicate your level of satisfaction with and the importance to you of each of the following characteristics of our product.

On a scale of 1 to 5, circle the appropriate number that indicates how satisfied you are with each of the following items. A score of 1 being very dissatisfied and 5 being very satisfied.

On a scale of 1 to 5, circle the appropriate number that indicates the importance to you of each of the following items. A score of 1 being very unimportant and 5 being very important (VI). Note that items 17 & 18 do not have importance scores since they are overall satisfaction items.

In the Comment section after each question, please include reasons for your satisfaction or dissatisfaction with this item including specific examples where possible.

Step	Satisfaction	Importance
1. Ease of installation of the software.	1 2 3 4 5	1 2 3 4 5
Comments:		
2. Completeness and accuracy of installation.	1 2 3 4 5	1 2 3 4 5
Comments:		
3. Ability of the initially delivered software to function without errors or problems.	1 2 3 4 5	1 2 3 4 5
Comments:		
4. Ability of the initially delivered software to function without crashes or service interruptions.	1 2 3 4 5	1 2 3 4 5
Comments:		

5. Long term ability of the software to function without errors or problems.	1 2 3 4 5	1 2 3 4 5
Comments:		
6. Long term ability of the software to function without crashes or service interruptions.	1 2 3 4 5	1 2 3 4 5
Comments:		
7. Ability of the user to easily perform required tasks using the software.	1 2 3 4 5	1 2 3 4 5
Comments:		
8. User friendliness of the software.	1 2 3 4 5	1 2 3 4 5
Comments:		
9. Completeness of the software in providing all of the functions I need to do my job.	1 2 3 4 5	1 2 3 4 5
Comments:		
10. Technical leadership of the functionality of this product compared to other similar products.	1 2 3 4 5	1 2 3 4 5
Comments:		
11. Availability of the technical support.	1 2 3 4 5	1 2 3 4 5
Comments:		
12 Ability of technical support to solve my problems.	1 2 3 4 5	1 2 3 4 5
Comments:		
13. Completeness of the user documentation.	1 2 3 4 5	1 2 3 4 5
Comments:		
14. Usefulness of the user documentation.	1 2 3 4 5	1 2 3 4 5
Comments:		
15. Ease of installation of the software.	1 2 3 4 5	1 2 3 4 5
Comments:		
16. Completeness of the training.	1 2 3 4 5	1 2 3 4 5
Comments:		

17. Overall, how satisfied are you with the Booking Books software product?	1 2 3 4 5	1 2 3 4 5
Comments:		
18. Overall, how satisfied are you with the Booking Books software products support services?	1 2 3 4 5	
Comments:		1 2 3 4 5

Guideline Application Tool

1. Description

All these general concepts (as general architecture, common tools, life cycle models, common document templates), which generally applies to all software development projects must be in *Twiki*, in a general section if applies to all projects, or in the development section of a project if only applies to one of them.

4.2.2. Quality manual

ISO 9001:2008 Standard

The organization shall establish and maintain a quality manual that includes

a) the scope of the quality management system, including details of and justification for any exclusions (see 1.2),

b) the documented procedures established for the quality management system, or reference to them, and

c) a description of the interaction between the processes of the quality management system.

ISO 90003:2004 Guideline

1. Description

This final career project can be understood as a quality manual.

Guideline Application Tool

1. Description

It is not necessary add any guideline application tool.

4.2.3. Control of Documents

ISO 9001:2008 Standard

Documents required by the quality management system shall be controlled. Records are a special type of document and shall be controlled according to the requirements given in 4.2.4.

A documented procedure shall be established to define the controls needed,

a) to approve documents for adequacy prior to issue,

b) to review and update as necessary and re-approve documents,

c) to ensure that changes and the current revision status of documents are identified,

d) to ensure that relevant versions of applicable documents are available at points of use,

e) to ensure that documents remain legible and readily identifiable,

f) to ensure that documents of external origin determined by the organization to be necessary for the planning and operation of the quality management system are identified and their distribution controlled, and

g) to prevent the unintended use of obsolete documents, and to apply suitable identification to them if they are retained for any purpose.

Guideline Application

1. Description

The objective of the control of documents is ensure that documents of the quality management system are prepared, reviewed, approved, published, distributed and managed according to the specifications in this procedure.

Applying this procedure to all documents generated internally or from external sources such as policies, regulations, standards, other normative documents, books, test methods, schemes (plans or drawings) software, specifications, instructions and manuals that are part of the system Quality and so on.

Terms and references:

- **Document.** Information and support medium.

- **Approval.** Status of document control procedure that determine the suitability and acceptance of documents.
- **External documents.** It is those documents that have an origin outside the organization.

2. Descriptions of the Stages of Internal Documents

Stage	Description	Responsible	Record
1 Elaboration	The content of the document is developed following the guidelines of the document and depending on the type of document to use.	Designated by the head of the process where applicable.	None.
2 Approval	It is reviewed and approved in order to be consistent with the reality of the company, and tailored to its needs.	Quality Committee.	Minutes of meeting.
3 Ensure that the appropriate version is available at point of use	Before publishing a document is placed a code. The documents produced are placed at points of use. Those that need to be distributed in printed form is placed a seal "Copy Controlled." In case you need a copy of any internal documents, you get a duplicate with the permission of the "Senior Management".	Quality coordinator.	None.
4 Release	Coordinator of quality management communicates to the processes responsible for the inclusion of new documents at points of use. Subsequently, and depending on the number of documents and its application is disclosed to staff or the owners of processes, for these staff to the Committee on Quality Improvement and communicate the document.	Responsible of the process in which the document applies.	Change request.
5 Revision	Changes in legislation.	Responsible of the process in which the document applies.	Change request.
6 State of the actual revision	The state of the actual version is identified with the version number and date placed in the header of the documents. Version starts with No. 1 and is increased when the document is updated. The date indicates from when the document comes into effect. Each time you build or modify an internal document is updated in the master list of internal documents.	Quality coordinator.	Master List of documents.
7. Identification of changes	From version 2 of the paper is added to the final table with the following information: Version No; Date;	Quality coordinator.	Control Table changes.

	Description of changes		
8 Update	Set the contents of the document according to the results of the review.	Designated by the head of the process in which applies/ Quality coordinator.	Change request.
9 Ensure that documents remain legible and readily identifiable	Documents are printed. Is maintained a master list of internal documents, where all internal documents related management system of quality. Is made a backup copy of these documents.	Quality coordinator.	Backup control.
10 control of obsolete documents	Obsolete copies of internal documents are destroyed by delivering the new version of the document and the new system replaces the previous version.	Quality coordinator.	None.

3. Description of the Stages for External Documents

Stage	**Description**	**Responsible**	**Record**
5.1 Acquisition	Once the external document is acquired, it is delivered to the process which required it.	Senior management/ responsible in which applies the Document/ Quality Coordinator.	Master List of external documents.
5.2 Provision of appropriate version at the points of use	Those responsible for each process are responsible of maintaining the external document in a protected and known place.	Responsible for the process in which the document applies.	None.
5.3 Release of the document.	The responsible of the process which applies the document, will inform to the people in his charge to handle the document.	Responsible of the process in which the document applies.	Minutes of meeting of the process where applicable.
5.4 Updates	Replace or append the text as shipments of updates or changes to the documents identified in the Master List external documents.	Allocated in the master list of external documents in the column "Responsible for the updating and distribution".	None.
5.5 Update status or current editing or distribution.	The updates sent of some documents are controlled through the standard for the control of the service. The documents which do not contain these sheets are controlled by the dates of its updates. Through the master list of external documents their distribution is controlled. Subsequently, the update or change is	Assigned to the "Master List of external documents" in the column Responsible for the updating and distribution.	Guideline for the control of service / date of the amendments.

	delivered to the responsible of the process.		
5.6 Control of obsolete documents	The documents are put on a stamped "Obsolete Document" and stored as prescribed in the "Master List of external documents."	Responsible of implement the process.	Obsolete documents with its seal.

Guideline Application Tool

1. Description

All the documents (internal and external) and its stages must be in a section in the Twiki.

4.2.4. Control of records

ISO 9001:2008 Standard

Records established to provide evidence of conformity to requirements and of the effective operation of the quality management system shall be controlled.

The organization shall establish a documented procedure to define the controls needed for the identification, storage, protection, retrieval, retention and disposition of records.

Records shall remain legible, readily identificable and retrievable.

4.2.4.1. Evidence of conformity to requirements

ISO 90003:2004 Guideline

Evidence of conformity to requirements may include

a) documented test results,

b) problem reports, including those related to tools problems,

c) change requests,

d) documents marked with comments,

e) audit and assessment reports, and

f) review and inspection records, such as those for design reviews, code inspections, and walk-through.

Guideline Application

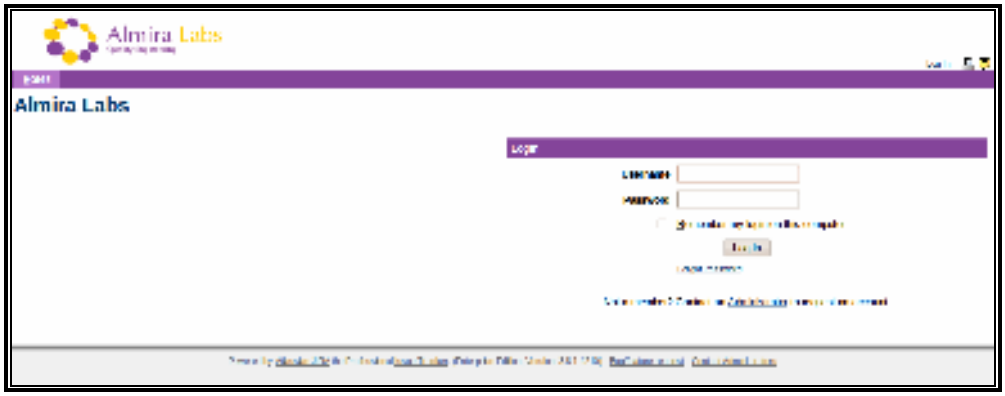
1. Description

The evidences of conformity to requirements elements are described in its own paragraph of this Project (the document test results, for example, are described in the test part); or can be described and collected with a *JIRA* issue (for example, change request, reviews, codes inspections, walk-through).

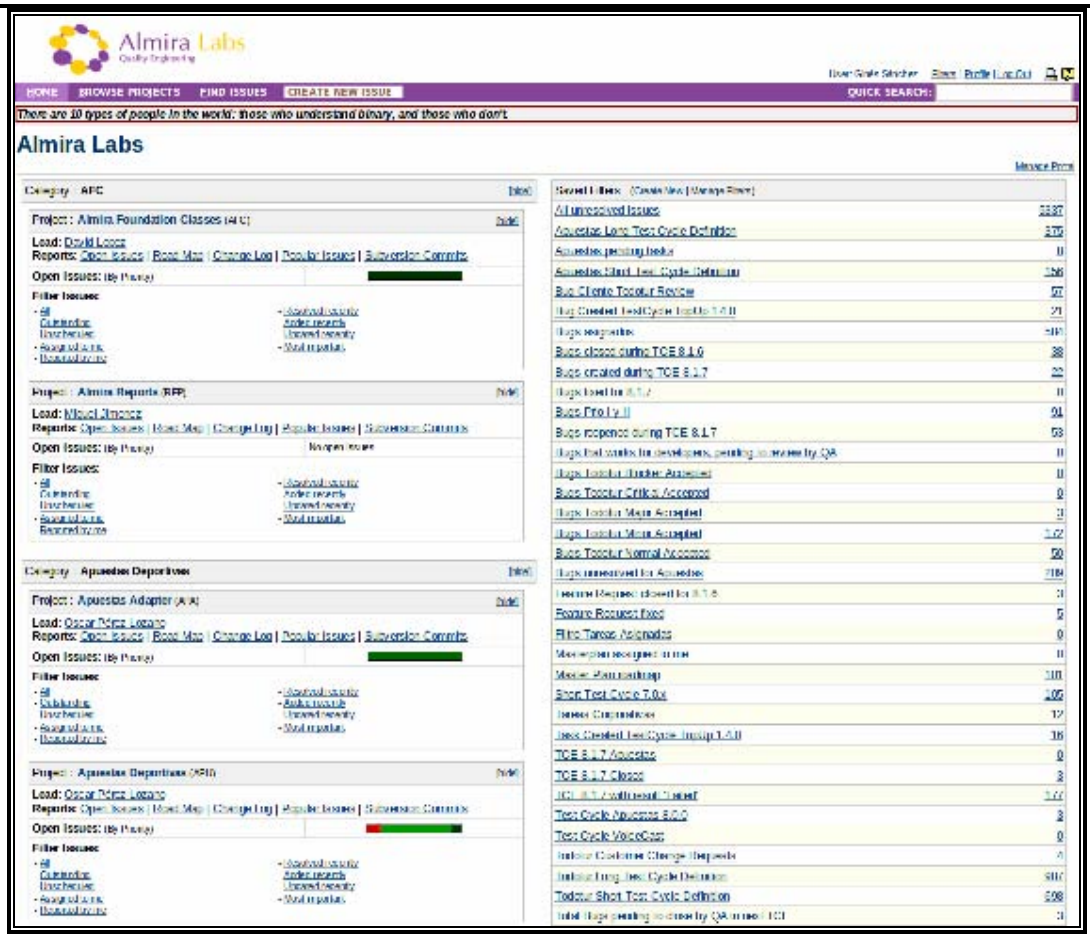
Guideline Application Tool

1. Description

In the next pictures, I describe how to create an issue in *Jira*. It can be a *Bug*, *Review*, *Change Request*, and so on. I simulate the process for create a bug; but user can create an issue of all the other types.

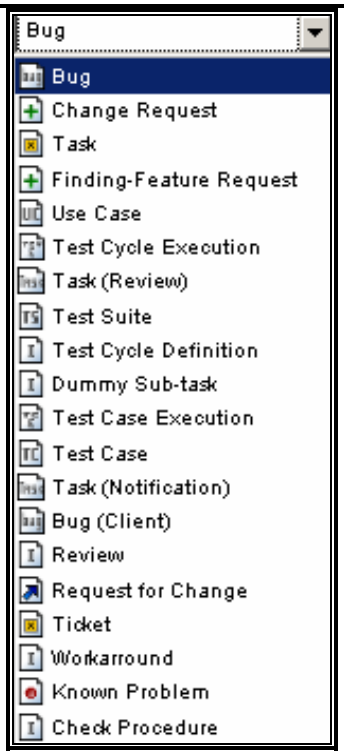


Picture III-30: Jira Authentication Page

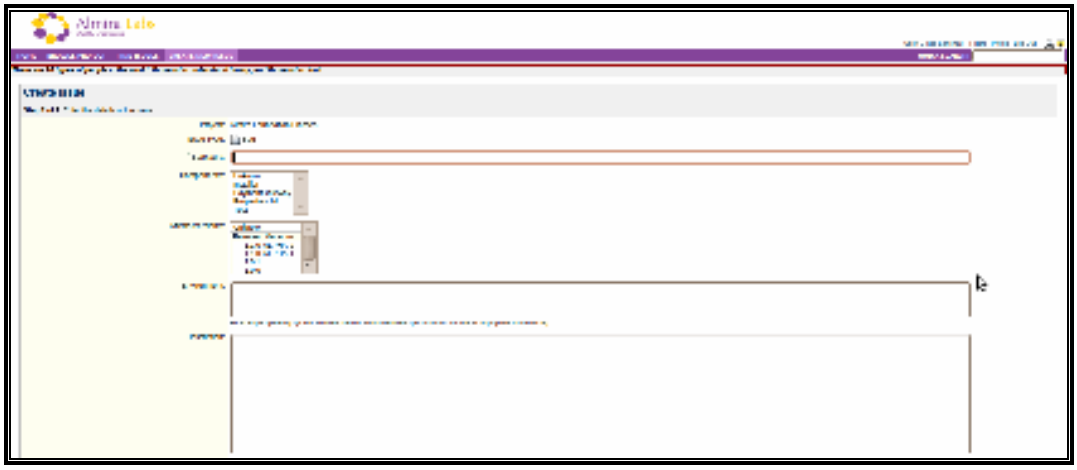


Picture III-31 Jira Main Page

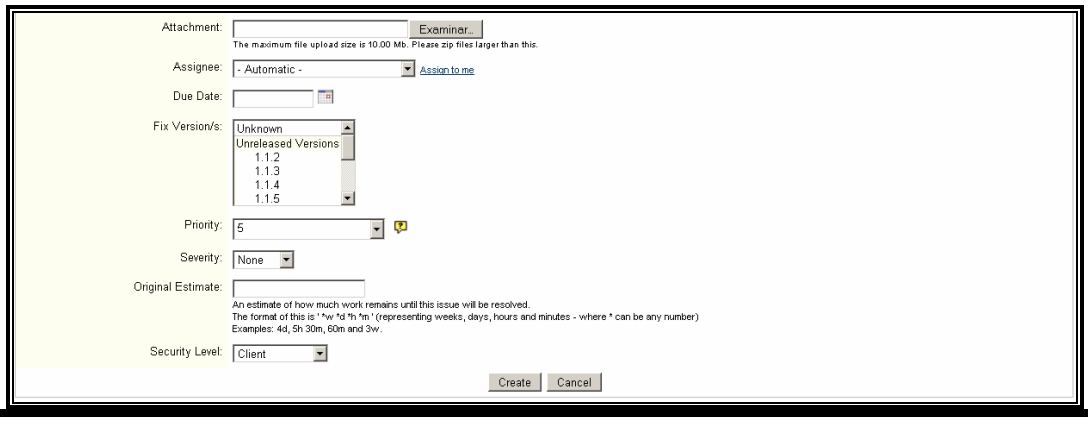
Picture III-32: Jira Issue Type Selection



Picture III-33: Jira Issue Type Selection Detail



Picture III-34: Jira Bug Summary Form



Picture III-35: Jira Bug Summary Form Detail

4.2.4.2. Evidence of effective operation

ISO 90003:2004 Guideline

Examples of evidence of effective operation of the quality management system may include, but are not limited to

a) changes (and the reasoning) to resources (people, software and equipment),

b) estimates, e.g. project size and effort (people, cost, schedule),

c) how and why tools, methodologies and suppliers were selected and qualified,

d) software license agreements (both for software supplied to customers and software procured to aid development),

e) minutes of meetings, and

f) software release records.

Guideline Application

1. Description

For the issues related to the evidence of effective operation of the quality management system, must be opened an issue to include all those examples (for example, when a minute of meeting is done an issue is created and assigned in order to be fulfilled). These issues must be reviewed and closed (resolved state) as soon as possible.

Guideline Application Tool

1. Description

For these issues, I will use Jira, because provide an issue tracking and project tracking for software development.

4.2.4.3. Retention and disposition

ISO 90003:2004 Guideline

When determining the retention periods for records, consideration should be given to statutory and regulatory requirements. Where records are held on electronic media, consideration of the retention times and accessibility of the records should take into account the rate of media degradation, the availability of the devices, and software needed to access the records. Records may include information held in email systems.

Protection from computer viruses and unapproved or illegal access should be considered.

The proprietary nature of the information stored on records should be assessed, in determining the methods of data erasure from the media, at the end of its required retention period.

NOTE For further general guidance related to ISO 9001:2000, 4.2, see ISO/IEC 12207:1995[11], 6.1, and ISO/IEC 12207:1995/Amd.1:2002[12], F.2.1 (documentation process).

Guideline Application

1. Description

All the records must be stored in the company server's infrastructure. Company must provide backup and anti-viruses protection. The retention period of records must be done accordingly to regulatory law and other legal restrictions (law restrictions are beyond the scope of this project).

Guideline Application Tool

1. Description

Records can be in *Jira*, *Twiki*, or in the software project documentation (*Xuse* xml files).

5. Management Responsibility

5.1. Management commitment

ISO 9001:2008 Standard

Top management shall provide evidence of its commitment to the development and implementation of the quality management system and continually improving its effectiveness by

a) communicating to the organization the importance of meeting customer as well as statutory and regulatory requirements,

b) establishing the quality policy,

c) ensuring that quality objectives are established,

d) conducting management reviews, and

e) ensuring the availability of resources.

Guideline Application

1. Description

It is not necessary add any guideline.

Guideline Application Tool

1. Description

Some parts of the management commitment can be reflected in *Twiki*, for example the quality policy.

5.2. Customer focus

ISO 9001:2004 Guideline

Top management shall ensure that customer requirements are determined and are met with the aim of enhancing customer satisfaction (see 7.2.1 and 8.2.1).

Guideline Application

1. Description

Customer requirements will be elicitated with the user stories and classified by its type. Not only top management shall ensure that customer requirements are met; project team leader, developers and all the staff have the responsibility of determined, implement and review those requirements for each use case.

Guideline Application Tool

1. Description

I will use *Xuse* to create, define and perform a traceability issues for every customer requirement from the user story and the use cases to the unit acceptance test.

5.3. Quality policy

ISO 9001:2008 Standard

Top management shall ensure that the quality policy

a) is appropriate to the purpose of the organization,

b) includes a commitment to comply with requirements and continually improve the effectiveness of the quality management system,

c) provides a framework for establishing and reviewing quality objectives,

d) is communicated and understood within the organization, and

e) is reviewed for continuing suitability.

Guideline Application

1. Description

All organizations have written some principles, doctrines, and beliefs that are the general guidelines of business conduct. In SME, one person tends to take all these decisions.

As ungrateful incidents are happening, there is an urgent need to write and publish a quality policy to guide the business activity. The main disadvantage of a written policy is the job entail, and the sense of limiting the scope to adapt to changing situations.

Top management must ensure to design reviews in order to meet market requirements and the regulations, both as regards their function and in terms of quality, quality, reliability and security.

Guideline Application Tool

1. Description

The quality policy must be published in the *Twiki*.

5.4. Planning

5.4.1. Quality objectives

ISO 9001:2008 Standard

Top management shall ensure that quality objectives, including those needed to meet requirements for product [see 7.1 a)], are established at relevant functions and levels within the organization. The quality objectives shall be measurable and consistent with the quality policy.

Guideline Application

1. Description

Quality Objectives, like all objectives should be smart objectives, that is, the objectives should be:

- **S**pecific
- **M**easurable
- **A**ttainable
- **R**elevant
- **T**imely

2. Quality Objectives

Critical Success Factors	Key Impact Processes	Key Performance Indicators	Source of KPI
Working proactively with customers	Client Activities- Account Management. Services– Implementation. Products – Planning.	Attendance to industry seminars and conferences relevant to our product portfolio (Minimum of 5 per year). Regular measurement of customer needs through customer satisfaction surveys. (Minimum of 2 surveys per year per product).	Training register. Customer Feedback Folder.
Improvement in Customer satisfaction	Client Activities– Support. Services– Implementation. Products – Development	Trend analysis of customer satisfaction survey results showing increase over time.	Customer Feedback Folder.
Responding to Customer needs	Client Activities – Sales. Client Activities– Support. Client Activities–Account Management. Products–Planning.	Evidence of being responsive to customer requests (bug fixes and enhancement). Quarterly review of software releases (Maintenance, Minor and Major releases).	Jira report.
Efficient software product development and maintenance	Product – Planning. Product – Development. Services-Custom Development.	Estimated development effort against actual effort (ratio of Timesheet report versus Project Plan).	Timesheet report, Project Plan.
Efficient provision of services	Services–Consultancy. Services– Implementation.	Profit gained, as calculated by the formula [Project Services Revenue] – ([Project Hours] x [Average Staff Hourly Rate]). Delivery on time and for the expected number of ratio person-hours.	Financials, Customer Feedback Folder.
Efficient use of resources	Services– Implementation. Products – Development.	Use rates (Minimum 80%). Total hours divided by DIV hours.	Use rate.
Providing Shareholder Value	Client Activities–Sales. Services–Implementation. Products–Development. Corporate Support– Finance.	Financials (Earnings per Share).	Financial accounts.
Culture of continuous improvement	Corporate Support- Business Process Improvement.	Execute preventive and corrective actions as suggested by staff and management (e.g. 20 per year).	Completed preventive and corrective action forms.

Committed and professional Staff	Corporate Support-Human Resources.	Staff turnover rate (Average tenure of 3 years).	Employment records.

Guideline Application Tool

1. Description

The quality objectives must be published in *Twiki*.

5.4.2. Quality management system planning

ISO 9001:2008 Standard

Top management shall ensure that

a) the planning of the quality management system is carried out in order to meet the requirements given in 4.1, as well as the quality objectives, and

b) the integrity of the quality management system is maintained when changes to the quality management system are planned and implemented.

ISO 90003:2004 Guideline

Planning may occur at organizational and project/product levels.

Quality management system planning at the organizational level may include the following:

a) defining appropriate software life cycle models to be used for the types of project that the organization undertakes, including how the organization normally implements software life cycle processes;

b) defining the work products of software development, such as software requirements documents, architectural design documents, detailed design documents, program code, and software user documentation;

c) defining the content of software management plans, such as software project management plans, software configuration management plans, software verification and validation plans, software quality assurance plans and training plans;

d) defining how software engineering methods are tailored for the organization's projects within the life cycle

e) identifying the tools and environment for software development, operations or maintenance;

f) specifying conventions for the use of programming languages, e.g. coding rules, software libraries and frameworks;

g) identifying any software reuse (see also 7.5.4).

The organization's management representative should consider any change to a software life cycle model which may affect the quality management system and should ensure that such changes do not compromise any quality management system controls.

Software quality planning at the project/product level is discussed in 7.1.

Guideline Application

1. Description

Planning is one of the basic concepts used to manage quality. Planning is one task during which establishes the objectives and develops the means to achieve them. Within the organization, quality plans are closely related. General quality objectives require general plans. These plans require sub objectives, each of which requires a sub plan.

All these elements have been defined in the step *4.2 Documentation Requirements*.

Guideline Application Tool

1. Description

The quality management system planning must be done in *Microsoft Project* and published in *Twiki*.

5.5. Responsibility, authority and communication

5.5.1. Responsibility and Authority

ISO 9001:2008 Standard

Top management shall ensure that responsibilities and authorities are defined and communicated within the organization.

Guideline Application

1. Description

The Authority is defined as an authority or power to do something. It is also the power of one person over another who is subordinate. And finally, means one or more people lined with some power or control. Authority within a firm is measured from the point of view of rank or title you hold within the organization.

The head of the company must fully define and communicate the duties to be assigned to their middle and intermediate, which in turn travel to their subordinates to carry out the proposed objectives. The duties for each person must be registered, making sure from the beginning to be understood by those involved.

Authorities Description has been defined in section *4.2.1.29. SME Authorities Description*.

Guideline Application Tool

1. Description

All these authorities descriptions must be published in a page named *Staff and Groups* or *Authorities Description* in *Twiki*.

5.5.2. Management representative

ISO 9001:2008 Standard

Top management shall appoint a member of management who, irrespective of other responsibilities, shall have responsibility and authority that includes

a) ensuring that processes needed for the quality management system are established, implemented and maintained,

b) reporting to top management on the performance of the quality management system and any need for improvement, and

c) ensuring the promotion of awareness of customer requirements throughout the organization.

NOTE: The responsibility of a management representative can include liaison with external parties on matters relating to the quality management system.

ISO 90003:2004 Guideline

For a software-producing organization, there is benefit if the management representative has had experience with software development.

Guideline Application

1. Description

In the *Authorities Description* defined in section *4.2.1.29. SME Authorities Description* can be found the roll CQO (Chief Quality Officer) with the responsibilities described for this paragraph.

Guideline Application Tool

1. Description

These authorities' descriptions must be published in a page named *Staff and Groups* or *Authorities Description* in *Twiki*.

5.5.3. Internal communication

ISO 90003:2004 Guideline

Top management shall ensure that appropriate communication processes are established within the organization and that communication takes place regarding the effectiveness of the quality management system.

Guideline Application

1. Description

ISO standards are a lot about communication. Management is supposed to communicate to employees the goals and vision of the company, specifics on the quality management system, and host of other subjects.

Internal communication can be done by each day, talking about different internal processes (fulfilment, projects, Internet marketing, quality, human resources, strategy, etc), and discussing the latest developments or current work happening on each. Employees can also celebrate each other's successes and talk about corrections or corrective actions that may result from defects or problems that arise.

That process can be a way to keep everyone of the company in the loop, see where they are, and compare that to where they would like to be. The important part is that the communication is happening.

Guideline Application Tool

1. Description

Communication must be done in regular time periods (by each day or week depending the issue type, for example), and the results must be published and reviewed creating an issue in *Jira*.

5.6. Management review

5.6.1. General

ISO 9001:2008 Standard

Top management shall review the organization's quality management system, at planned intervals, to ensure its continuing suitability, adequacy and effectiveness. This review shall include assessing opportunities for improvement and the need for changes to the quality management system, including the quality policy and quality objectives.

Records from management reviews shall be maintained (see 4.2.4).

Guideline Application

1. Description

In most situations, which is supposed to be a planned and systematic review of the quality management system becomes a unique annual event. These meetings often require many hours to review all points of the system of quality management.

The solution is to establish a planning review with a high frequency. This frequency should be high enough to say that we are conducting a continuous review (monthly or every fifteen days, for example).

To set the frequency to revise the performance evaluation process will take into account the importance of each process in our organization and the possibility of changing these over time; it will depend of the process type: *Strategic Processes*, *Key Processes* or *Support Process*.

Guideline Application Tool

1. Description

Quality management system review must be done creating an issue in *Jira*.

5.6.2. Review input

ISO 9001:2008 Standard

The input to management review shall include information on

a) results of audits,

b) customer feedback,

c) process performance and product conformity,

d) status of preventive and corrective actions,

e) follow-up actions from previous management reviews,

f) changes that could affect the quality management system, and

g) recommendations for improvement.

ISO 90003:2004 Guideline

Guidance is provided for ISO 9001:2008, 5.6.2, item c) as follows.

One way to measure process performance is to perform software process assessments (see 8.2.3). The outcomes of software process assessments should be considered as input to management reviews.

One way to measure product conformity is to perform software product evaluation (see 8.2.4). The outcomes of software product evaluation should be considered as input to management review.

Guideline Application

1. Description

It is not necessary add any guideline.

Guideline Application Tool

1. Description

It is not necessary add any guideline application tool.

5.6.3. Review output

ISO 9001:2008 Standard

The output from the management review shall include any decisions and actions related to

a) improvement of the effectiveness of the quality management system and its processes,

b) improvement of product related to customer requirements, and

c) resource needs.

Guideline Application

1. Description

It is not necessary add any guideline.

Guideline Application Tool

1. Description

It is not necessary add any guideline application tool.

6. Resources Management

6.1 Provision of resources

ISO 9001:2008 Standard

The organization shall determine and provide the resources needed

a) to implement and maintain the quality management system and continually improve its effectiveness, and

b) to enhance customer satisfaction by meeting customer requirements.

Guideline Application

1. Description

The biggest failure of top management is not provide the resources necessary for the establishment and maintenance of quality management system.

The necessary resources are related to the actions added to subordinates, such as the time required to develop an action. Resources are a price to be paid to achieve the objectives. If not supplied, the lowest levels seen that these objectives do not have the priority that others to those that have been assigned.

Guideline Application Tool

1. Description

It is not necessary add any guideline application tool.

6.2 Human resources

6.2.1 General

ISO 9001:2008 Standard

Personal performing work affecting conformity to product requirements shall be competent on the basis of appropriate education, training, skills and experience.

NOTE Conformity to product requirements can be affected directly or indirectly by personnel performing any task within the quality managements system.

Guideline Application

1. Description

Human resource describes the individuals who comprise the workforce of an organization. In order to select the human resources with the adequate awareness and training, I have described a recruitment template in section *4.2.1.27. Recruitment Interview Template*.

Guideline Application Tool

1. Description

Recruitment Interview Template must be published in *Twiki*.

6.2.2 Competence, awareness and training.

ISO 9001:2008 Standard

The organization shall

a) determine the necessary competence for personnel performing work affecting product requirements,

b) where applicable, provide training or take other actions to achieve the necessary competence,

c) evaluate the effectiveness of the actions taken,

d) ensure that its personnel are aware of the relevance and importance of their activities and how they contribute to the achievement of the quality objectives, and

e) maintain appropriate records of education, training, skills and experience (see 4.2.4).

ISO 90003:2004 Guideline

The training needs should be determined considering the requirements notation, design methods, specific programming languages, tools, techniques and computer resources to be used in the development and management of the software product/project. It might also be useful to include training in the skills and knowledge of the specific field within which the software is applied and in other topics such as project management.

The technologies employed in software development, operation and maintenance should be continually monitored and evaluated in order to determine requirements for updating staff skills.

The form of training may not be necessarily traditional training courses but could be workshops, computer based training, self-study, mentoring, training on-the-job or web-based training.

Evaluation of the effectiveness of training may be performed using measurements of products and processes, identifying areas of improvement in personal performance (among other areas for improvement).

Guideline Application

1. Description

People fill many different positions in a software development organization. Preparing them for these positions takes education and pin-pointed training followed by careful assignment supported by mentoring and helpful supervision. An organization faces a substantial task when it moves a person from a latent *resource* to a particular position as a worker.

A worker is a position which may be assigned and which they accept. A worker type is a role that an individual may play in software development, such as use-case specifier, architect, component engineer, and integration tester.

Each worker is responsible for a whole set of activities, such as the activities involved in the design of a subsystem. To work effectively, workers need the information that is required to carry out those activities. The need to understand what their roles are relative to those of other workers. At the same time, if they are to do their work, the tools they employee must be adequate.

A worker must also be realized as a set of individuals working together. For example, an architect worker may be realized as an architecture board.

Each worker has a set of responsibilities and performs a set of activities in developing software. When allocating resources to workers in a project, the project manager needs to identify the

competencies of individuals and match them with the required competencies of the workers. The skills of the resources must be matched against the competencies specified by the various workers needed by the project.

Guideline Application Tool

1. Description

It is not necessary add any guideline application tool.

6.3 Infrastructure

ISO 9001:2008 Standard

The organization shall determine, provide and maintain the infrastructure needed to achieve conformity to product requirements. Infrastructure includes, as applicable

a) buildings, workspace and associated utilities,

b) process equipment (both hardware and software), and

c) supporting services (such as transport, communication or information systems).

ISO 90003:2004 Guideline

The infrastructure should include hardware, software, tools and facilities for development, operation or maintenance of software.

The infrastructure may include software tools that support the design and development process including the following:

a) tools, such as for analysis, design and development, configuration management, testing, project management, documentation, code creation or generation;

b) application development and support environments;

c) knowledge management, intranet, extranet tools;

d) network tools, including security, backup, virus protection, firewall;

e) help desk and maintenance tools;

f) access controls;

g) software libraries;

h) operations control tools such as for network monitoring, systems management and storage management.

Whether these tools and techniques are developed internally or are purchased, the organization should evaluate whether or not they are fit for purpose. Tools used in the implementation of the product, such as analysis and design and development tools, compilers and assemblers should be evaluated, approved and placed under an appropriate level of configuration management control prior to use. The scope of use of such tools and techniques may be documented with appropriate guidance, and their use reviewed, as appropriate, to determine whether there is a need to improve and/or upgrade them.

NOTE For further information, see the following:

— ISO/IEC 12207:1995[11], 7.2 and ISO/IEC 12207:1995/Amd.1:2002[12], F.3.2 (infrastructure process);

— ISO/IEC 14598-2[14] (acquisition) and ISO/IEC 14598-3[15] (evaluation of a software product);

— ISO/IEC 14102[13].

Guideline Application

1. Description

In the step *4.2.1. General documentation requirements* I have explained some tools used to develop this project.

Could be necessary add some other server infrastructure, for example to work on Virtual Machines, guides for create some users in a LDAP server, create new projects, maintain the company mail server, backup tools, server security, servers hosting and so on. All these elements depend of each company, and therefore I will not refer to them.

Guideline Application Tool

1. Description

The entire company infrastructure must be published in *Twiki*.

6. 4 Work environment

ISO 9001:2008 Standard

The organization shall determine and manage the work environment needed to achieve conformity to product requirements.

NOTE The term "work environment" relates to those conditions under which work is performed including physical, environmental and other factors (such as noise, temperature, humidity, lighting or whether).

Guideline Application

1. Description

The work environment is comprised of the physical location, equipment, materials processed or used, and the activities of an employee while engaged in the performance of his work, whether on or off the railroads property.

In order to have a work environment that improves the productivity of a software team, the company should meet the next requirements.

2. Features

2.1. Work Environment for Software Developers

- Must have two or three computer screens for the main computer (this could be replaced with virtual desktops).
- Must have a second computer on which to run batch processes or install things unstable.
- Must being surrounded by slates, and have a slate to put post-it.
- The work environment must have a low noise level and a correct acclimatization.
- Dimensions of the job place must be correct: not to work locked in a cubicle or office, or in a massed office.
- The worker computer must have all the necessary connections to work.
- Table and chair must provide a good posture, allowing to move the legs leave freely, to change position.

• The lighting, if white light and a well-lit, trying to match the brightness of the screen with the brightness of the office to avoid creating a lot of contrast.

• Avoid places of transit; workers should not have visual distractions.

• Have a space on one side needed to have manuals, reference books, magazines or specifications.

• Printer, copier, and scanner are always necessary.

2.2. Company Work Environment

• A room with their network and power points to improvise:

- Conference room for some external consultant

- Working room, for mount and dismount some hardware

- Screening room, with a video projector and a screen

• A room for relaxation, coffee machine, food, and so on; keep people outside the office helps to improve and create a corporate culture.

Guideline Application Tool

1. Description

It is not necessary add any guideline application tool.

7. Product Realization

7.1 Planning of product realization

ISO 9001:2008 Standard

The organization shall plan and develop the processes needed for product realization. Planning of product realization shall be consistent with the requirements of the other processes of the quality management system (see 4.1).

In planning product realization, the organization shall determine the following, as appropriate:

a) quality objectives and requirements for the product;

b) the need to establish processes and documents, and to provide resources specific to the product;

c) required verification, validation, monitoring, measurement, inspection and test activities specific to the product and the criteria for product acceptance;

d) records needed to provide evidence that the realization processes and resulting product meet requirements.

The output of this planning shall be in a form suitable for the organization's method of operations.

NOTE 1 A document specifying the processes of the quality management system (including the product realization processes) and the resources to be applied to a specific product, project or contract can be referred to as a quality plan.

NOTE 2 The organization may also apply the requirements given in 7.3 to the development of product realization processes.

Guideline Application

1. Description

The purpose of this project plan is create an online library called *Booking Books* developed in *J2EE*.

All the project plan sections described in section *4.2.1.25. Quality Plan Document Template* can be fulfilled following the guidelines of this project. For example, *2.1.3 Definitions,*

acronyms, and abbreviations are in the section *3. Terms and Definitions*.

7.1.1 Software life cycle

ISO 90003:2004 Guideline

Processes, activities and tasks should be planned and performed using life cycle models suitable to the nature of a software project, considering size, complexity, safety, risk and integrity. ISO 9001:2008 is intended for application irrespective of the life cycle models used and is not intended to indicate a specific life cycle model or process sequence.

Design and development can be an evolutionary process and procedures may therefore need to be changed or updated as the project progresses, after consideration of changes to related activities and tasks.

Consideration should be given to the suitability of the design and development method for the type of task, product or project and the compatibility of the application, the methods and the tools to be used. For products where failure may cause injury or danger to people, or damage or corruption of property or the environment, design and development of such software should ensure definition of specific design and development requirements that specify desired immunity from, and response to, potential failure conditions.

Software development planning should result in a definition of what products are to be produced, who is to produce them, and when they are to be produced (see 7.3.1). Software quality planning at the project/product level should result in a description of how specific products are to be developed, assessed or maintained.

Guideline Application

1. Description

The software life cycle used in this project is described in section *4.2.1.23. USDP Software Life Cycle*.

2. Booking Books Software Iterations

In the next pictures I have created a planning for *Booking Books* web application. It is composed of four development iterations and a final iteration with a *Final Test Cycle*, a *Production Environment Test* and a *Unit Acceptance Test*.

Guideline Application Tool

1. Description

Id	Nombre de tarea	Duración	Comienzo	Fin	Predece
1					
2	**ITERATION0**	**5días**	**mié01/09/10**	**mar07/09/10**	
3	User Stories Specification	2días	mié01/09/10	jue02/09/10	
4	Requirements Specification	3días	vie03/09/10	mar07/09/10	3
5					
6					
7	**ITERATION1**	**10días?**	**mié08/09/10**	**mar21/09/10**	**2**
8	**Development Activities**	**5días**	**mié08/09/10**	**mar14/09/10**	
9	UC_001 Customer Registration Specification	1día	mié08/09/10	mié08/09/10	
10	UC_001 Customer Registration Implementation	4días	jue09/09/10	mar14/09/10	9
11	**QAActivities**	**5días?**	**mié15/09/10**	**mar21/09/10**	**8**
12	UC_001 Test Case Specification	1día?	mié15/09/10	mié15/09/10	
13	UC_001 Test Case Automation	4días	jue16/09/10	mar21/09/10	12
14					
15					
16	**ITERATION2**	**10días?**	**mié15/09/10**	**mar28/09/10**	
17	**Development Activities**	**5días?**	**mié15/09/10**	**mar21/09/10**	**8**
18	UC_002 Customer Password Recovery Specification	1día?	mié15/09/10	mié15/09/10	
19	UC_002 Customer Password Recovery Implementation	4días	jue16/09/10	mar21/09/10	18
20	**QAActivities**	**5días?**	**mié22/09/10**	**mar28/09/10**	**17;11**
21	UC_002 Test Case Specification	1día?	mié22/09/10	mié22/09/10	
22	UC_002 Test Case Automation	4días	jue23/09/10	mar28/09/10	21
23					
24					
25	**ITERATION3**	**11días?**	**mié22/09/10**	**mié06/10/10**	
26	**Development Activities**	**6días?**	**mié22/09/10**	**mié29/09/10**	**17**
27	UC_003 Customer Account Modification Specification	1día?	mié22/09/10	mié22/09/10	
28	UC_003 Customer Account Implementation	5días	jue23/09/10	mié29/09/10	27
29	**QAActivities**	**5días?**	**jue30/09/10**	**mié06/10/10**	**26;20**
30	UC_003 Test Case Specification	1día?	jue30/09/10	jue30/09/10	
31	UC_003 Test Case Automation	4días	vie01/10/10	mié06/10/10	30
32					
33					
34	**ITERATION4**	**10días?**	**jue30/09/10**	**mié13/10/10**	
35	**Development Activities**	**5días?**	**jue30/09/10**	**mié06/10/10**	**26**
36	UC_004 Customer Books Search Specification	1día?	jue30/09/10	jue30/09/10	
37	UC_004 Customer Books Search Implementation	4días	vie01/10/10	mié06/10/10	36
38	**QAActivities**	**5días?**	**jue07/10/10**	**mié13/10/10**	**35;29**
39	UC_004 Test Case Specification	1día?	jue07/10/10	jue07/10/10	
40	UC_004 Test Case Automation	4días	vie08/10/10	mié13/10/10	39
41					
42					

Proyecto: Proyect1
Fecha: mar 31/08/10

Tarea — Resumen del proyecto
División — Tareas externas
Progreso — Hito externo
Hito — Fecha límite
Resumen

Página1

Picture III-36: Microsoft Project Gantt Page (I)

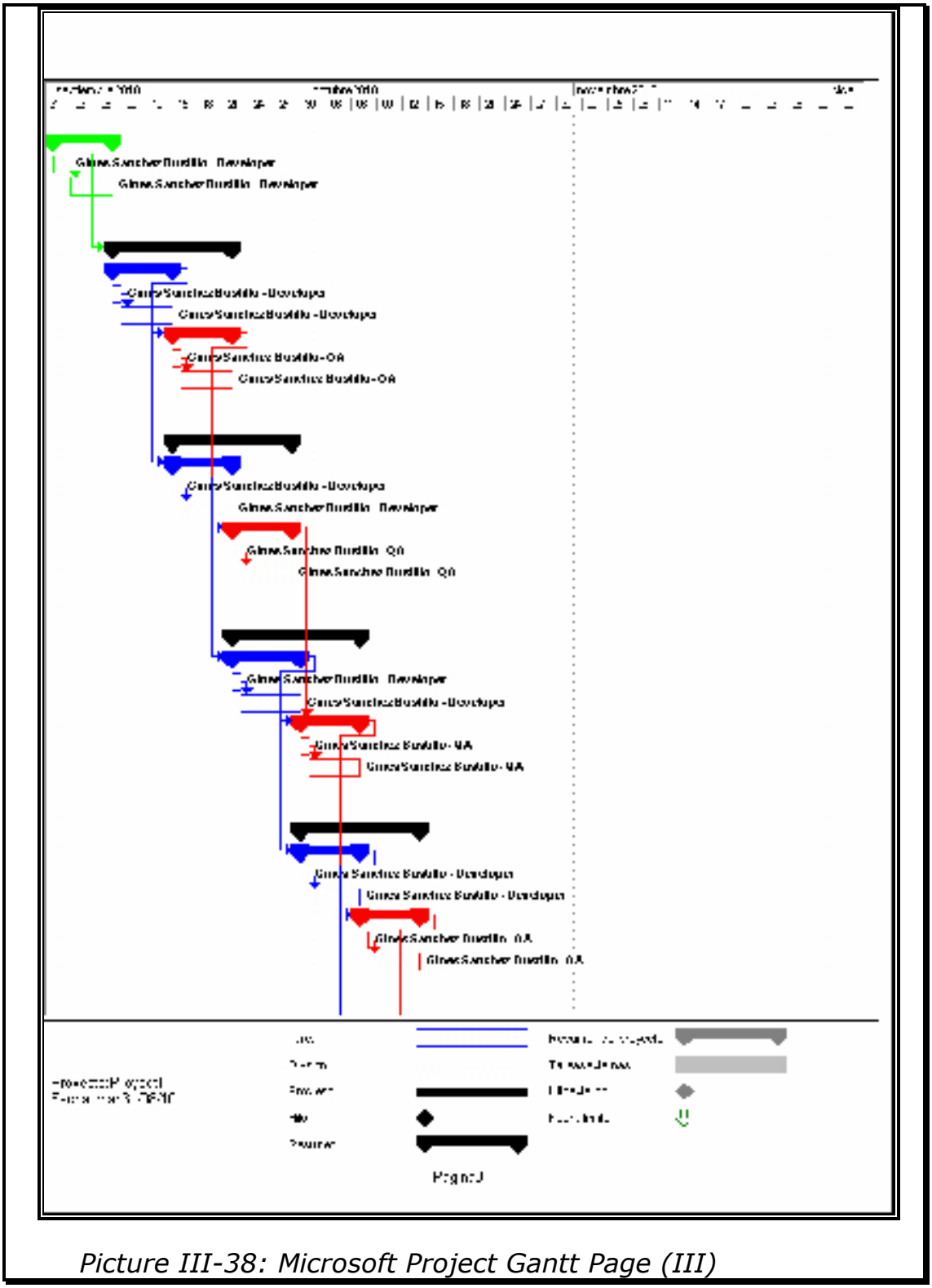
Id	Nombre de tarea	Duración	Comienzo	Fin	Predece
43	**ITERATION5**	**23días**	**jue07/10/10**	**lun08/11/10**	
44	**Development Activities**	**5días**	**jue07/10/10**	**mié13/10/10**	**35**
45	UC_005 Customer Login Specification	1día	jue07/10/10	jue07/10/10	
46	UC_005 Customer Login Implementation	4días	vie08/10/10	mié13/10/10	45
47	**QAActivities**	**18días**	**jue14/10/10**	**lun08/11/10**	**44;38**
48	UC_005 Test Case Specification	1día	jue14/10/10	jue14/10/10	
49	UC_005 Test Case Automation	4días	vie15/10/10	mié20/10/10	48
50					
51					
52	**ACCEPTANCE ITERATION**	**13días**	**jue21/10/10**	**lun08/11/10**	
53	Final Test Cycle	8días	jue21/10/10	lun01/11/10	49
54	Production Environment Test	3días	mar02/11/10	jue04/11/10	53
55	Unit Acceptance Test	2días	vie05/11/10	lun08/11/10	54

Proyecto: Proyect1
Fecha: mar 31/08/10

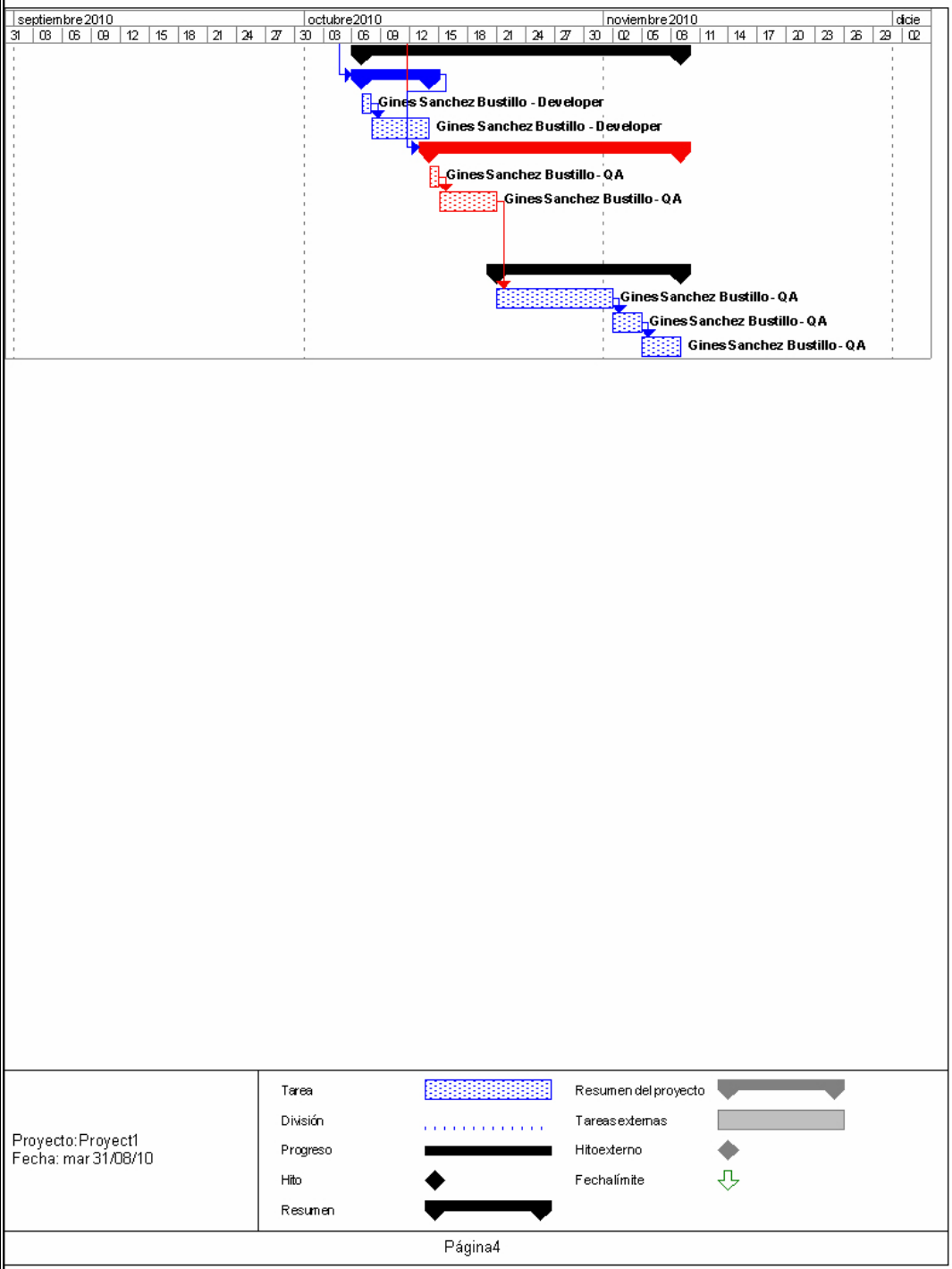
Tarea
División
Progreso
Hito
Resumen
Resumen del proyecto
Tareas externas
Hito externo
Fecha límite

Página2

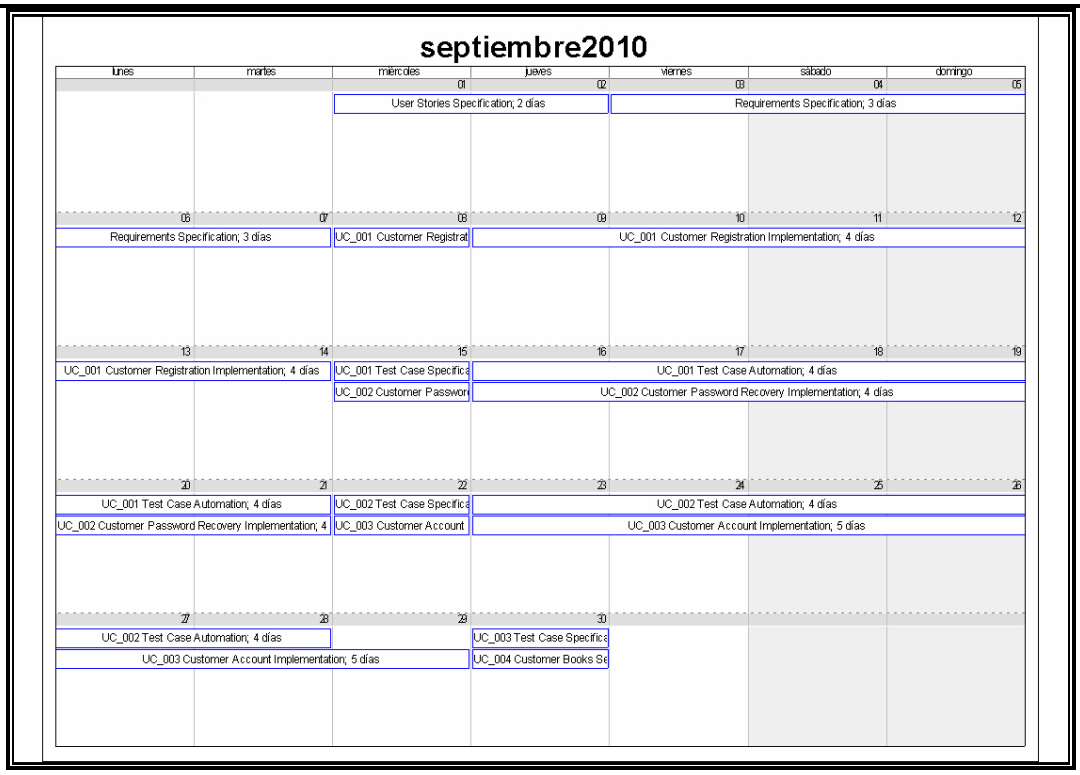
Picture III-37: Microsoft Project Gantt Page (II)



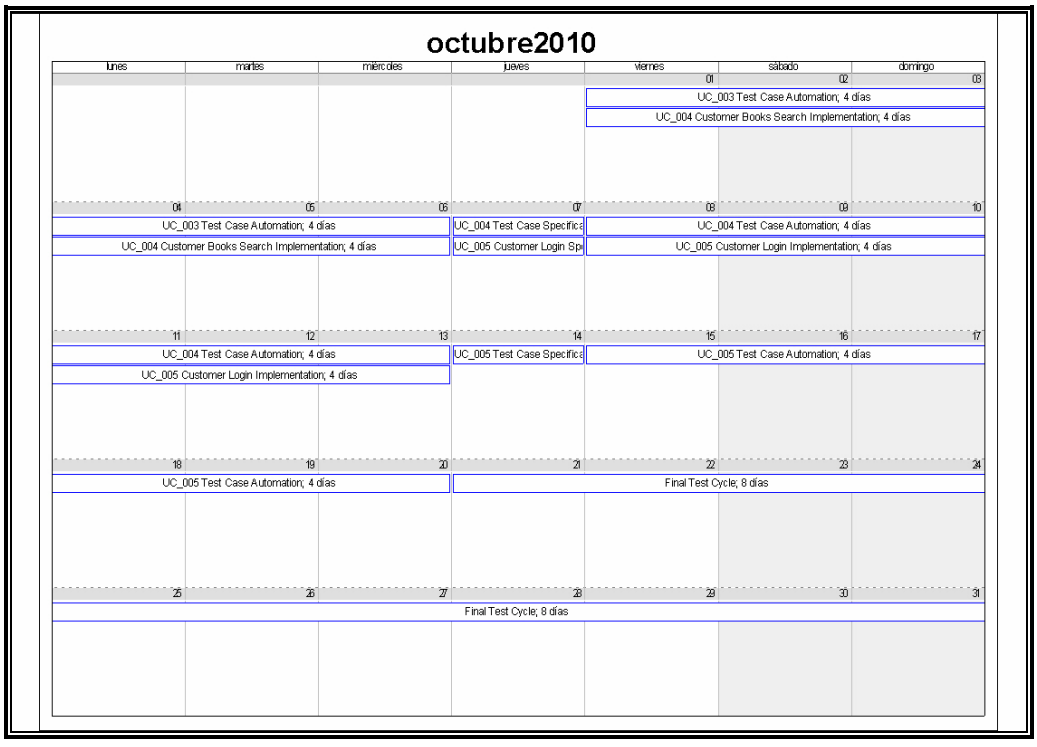
Picture III-38: Microsoft Project Gantt Page (III)



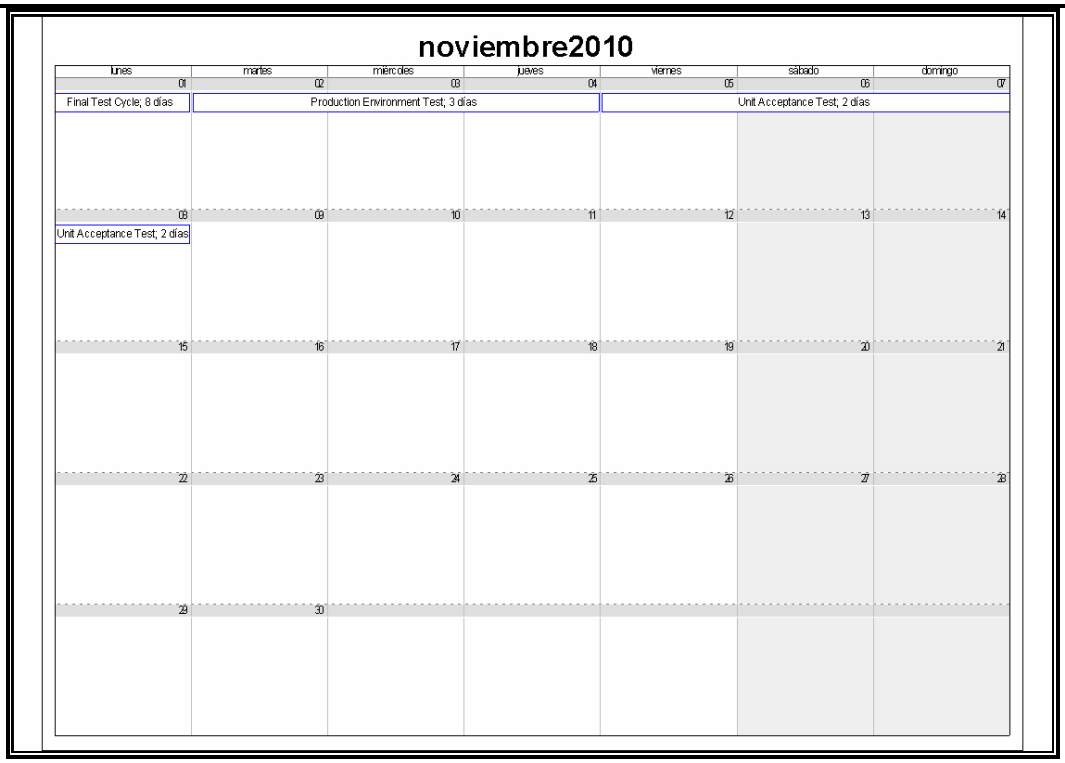
Picture III-39: Microsoft Project Gantt Page (IV)



Picture III-40: Microsoft Project Calendar Page (I)



Picture III-41: Microsoft Project Calendar Page (II)



Picture III-42: Microsoft Project Calendar Page (III)

7.1.2 Quality planning

ISO 90003:2004 Guideline

Quality planning provides the means for tailoring the application of the quality management system to a specific project, product or contract. Quality planning may include or reference generic and/or project/product procedures, as appropriate. Quality planning should be re-visited along with the progress of design and development, and items concerned with each stage should be completely defined when starting that stage.

Quality planning may be reviewed and agreed by all organizations concerned in its implementation, as appropriate.

Software quality planning at the project level should address the following:

a) inclusion of, or reference to, the plans for development (see 7.3.1);

b) quality requirements related to the product and/or processes;

c) quality management system tailoring and/or identification of specific procedures and instructions, appropriate to the scope of the quality manual and any stated exclusions (ISO 9001:2008, 1.2);

d) project-specific procedures and instructions, such as software test specifications detailing plans, designs, test cases and procedures for unit, integration, system and acceptance testing (see 8.2.4);

e) methods, life cycle model(s), tools, programming language conventions, libraries, frameworks and other reusable assets to be used in the project;

f) criteria for starting and ending each project stage;

g) types of review, and other verification and validation activities to be carried out (see 7.3.4, 7.3.5 and 7.3.6);

h) configuration management procedures to be carried out;

i) monitoring and measurement activities to be carried out;

j) the person(s) responsible for approving the outputs of processes for subsequent use;

k) training needs in the use of tools and techniques, and scheduling of the training before the skill is needed;

l) records to be maintained (see 4.2.4);

m) change management, such as for resources, timescale and contract changes.

Quality planning, however abbreviated, is particularly useful to clarify limited quality objectives for software being designed for a limited purpose. Examples of limited-purpose software include proof-of-concept demonstration prototypes, a research computation used only by its designer, an interim solution lacking features such as security or full operational performance that will be implemented in a future output, and onetime data analysis reports.

Limited-purpose software should be tested in ways that are consistent with its planned use to reduce the possible occurrence of unintended omissions and errors.

NOTE 3 For further general guidance related to ISO 9001:2000, 7.1, see the following:

— ISO/IEC 12207:1995[11], 5.2.4 (planning), 5.3.1 (development process implementation), and 6.1 to 6.8, and

ISO/IEC 12207:1995/Amd.1:2002[12], F.2 (supporting life cycle processes);

— ISO/IEC 9126-1:2001[5];

— ISO/IEC 14598-2[14];

— ISO/IEC TR 15846:1998[27], 6.2 (planning of configuration management);

— ISO/IEC TR 16326:1999[30], 6.2.2 (planning of project management).

Guideline Application

1. Description

All the software quality plan sections described in section *4.2.1.25. Quality Plan Document Template* can be fulfilled following the guidelines of this project.

For example, in section *5. Standards, Practices, Conventions, and Metrics* is in the section *4.2. Documentation Requirements*; or *7. Tests* is in the section *7.3.6.2. Testing*.

Guideline Application Tool

1. Description

Guideline application tools are described for each section.

7.2 Customer-related processes

7.2.1 Determination of requirements related to the product

ISO 9001:2008 Standard

The organization shall determine

a) requirements specified by the customer, including the requirements for delivery and post-delivery activities,

b) requirements not stated by the customer but necessary for specified or intended use, where known,

c) statutory and regulatory requirements related to the product, and

d) any additional requirements determined by the organization.

NOTE Post-delivery activities include, for example, actions under warranty provisions, contractual obligations such as maintenance services, and supplementary services such as recycling or final disposal.

7.2.1.1 Customer-related requirements

ISO 90003:2004 Guideline

Software may be developed as part of a contract, as a product available for a market sector, as software embedded in a system or in

support of the business processes of the organization. Requirements determination is applicable in all these circumstances.

Specific actions may include:

a) the establishment of the following for developing the requirements:

1) methods for agreement of requirements and authorizing and tracking changes, especially during iterative development;

2) methods for the evaluation of prototypes or demonstrations, where used;

3) methods for recording and reviewing discussion results from all parties involved;

b) the development of the requirements in close co-operation with the customer or users, and efforts to prevent misunderstandings by, for example, the provision of definition of terms, explanation of the background of requirements;

c) the obtention of the customer's approval of the requirements;

d) the establishment of a method for traceability of the requirements to the final product (such as a requirements traceability matrix).

The requirements may be provided by the customer, may be developed by the organization or may be jointly developed.

When the requirements are provided and agreed in the form of a system specification, methods should be in place to allocate them into hardware and software items with any appropriate interface specifications. Changes to the requirements should be controlled. The contract may need to be amended when requirements change. In contractual situations, the requirements may not be fully defined at contract acceptance, and some may be developed during a project.

The requirements may need to take the operational environment into account. The requirements may include, but not be limited to, the following characteristics: functionality, reliability, usability, efficiency, maintainability and portability. Other characteristics may be specified, for example security, safety and statutory obligations. Some of these characteristics may be mission and/or safety critical.

If the software product needs to interface with other software or system products, the interfaces between the software product to be developed and other software or system products should be specified, as far as possible, either directly or by reference, in the requirements.

The requirements should be expressed in clear and unambiguous terms that facilitate validation during product acceptance. Requirements should be traceable throughout the development life cycle (see 7.5.3)

Guideline Application

1. Description

Requirements related to the product are described in section *7.3.3. Design and development outputs*.

Guideline Application Tool

1. Description

Guideline application tools are described for each section.

7.2.1.2 Additional requirements determined by the organization

ISO 90003:2004 Guideline

NOTE 1 For further information on 7.2.1.1, see the following:

— ISO/IEC 12207:1995[11], 5.3.2 to 5.3.4 (development process) and ISO/IEC 12207:1995/Amd.1:2002[12], F.1.3.1

(requirements elicitation), F.1.3.2 (systems requirements analysis) and F.1.3.4 (software requirements analysis);

— ISO/IEC 9126-1:2001[5];

— ISO/IEC 15026:1998[20].

NOTE 2 For further information on 7.2.1.2, see ISO/IEC 12119:1994[10].

Guideline Application

1. Description

The organization's requirements should be defined in technical manuals that are used by designers, production and all the other staff. These will often apply to all the organization's product and service but will, however, need to be reviewed to identify the specific requirements that apply to particular products.

Guideline Application Tool

1. Description

Additional requirements must be published in *Twiki*.

7.2.2 Review of requirements related to the product

ISO 9001:2008 Standard

The organization shall review the requirements related to the product. This review shall be conducted prior to the organization's commitment to supply a product to the customer (e.g. submission of tenders, acceptance of contracts or orders, acceptance of changes to contracts or orders) and shall ensure that

a) product requirements are defined,

b) contract or order requirements differing from those previously expressed are resolved, and

c) the organization has the ability to meet the defined requirements.

Records of the results of the review and actions arising from the review shall be maintained (see 4.2.4).

Where the customer provides no documented statement of requirement, the customer requirements shall be confirmed by the organization before acceptance.

Where product requirements are changed, the organization shall ensure that relevant documents are amended and that relevant personnel are made aware of the changed requirements.

NOTE In some situations, such as internal sales, a formal review is impractical for each order. Instead the review can cover relevant product information such as catalogues or advertising material.

7.2.2.1 Organization concerns

ISO 90003:2004 Guideline

Issues which may be relevant during the organization’s review of software tenders, contracts or orders include, but are not limited to the following:

a) the feasibility of meeting and validating the requirements and product characteristics, including identifying the required software characteristics (e.g. functionality, reliability, usability, maintainability, portability and efficiency);

b) software design and development standards and procedures to be used;

c) the identification of facilities, tools, software items and data, to be provided by the customer, the definition and documentation of methods to assess their suitability for use;

d) the operating system or hardware platform;

e) agreement on the control of external interfaces with the software product;

f) replication and distribution requirements;

g) customer-related issues:

1) life cycle processes imposed by the customer;

2) period of obligation of the organization to supply copies and the capability of reading master copies;

h) management issues:

1) risk management should be addressed (see also 7.2.2.2);

2) organization's responsibility with regard to subcontracted work;

3) scheduling of progress, technical reviews and outputs;

4) installation, maintenance and support requirements;

5) timely availability of technical, human and financial resources;

i) legal, security and confidentiality issues:

1) information handled under the contract may be subject to concerns regarding intellectual property rights, license agreements, statutory and regulatory requirements, confidentiality and the protection of information including patents and copyrights;

2) guardianship of the master copy of the product and the rights of the customer to access or verify that master;

3) level of information disclosure, to the customer, needs to be mutually agreed to by the parties;

4) definition of warranty terms;

5) liabilities/penalties associated with the contract.

Guideline Application

1. Description

It is not necessary add any guideline.

Guideline Application Tool

1. Description

It is not necessary add any guideline application tool.

7.2.2.2 Risks

ISO 90003:2004 Guideline

The following risks may be included when reviewing requirements related to the product:

a) criticality, safety and security issues;

b) capabilities and experience of the organization or its suppliers;

c) reliability of estimates of resources and the duration required for each activity;

d) significant differences between the times required to deliver products or services, and the times determined from plans through the optimization of cost and quality goals;

e) significant geographical dispersion of the organization, customers, users and suppliers;

f) high technical novelty, including novel methods, tools, technologies and supplied software;

g) low quality or availability of supplied software and tools;

h) low precision, accuracy and stability of the definition of the customer requirements and external interfaces.

The implications of any contract changes on resources, schedules and costs should be evaluated, particularly for changes to scope, functionality or risk. The above issues should be re-evaluated, as appropriate.

Guideline Application

1. Description

Model for risk management can be found in section *4.2.1.26. Risk Management Document Template*.

Software risk management is quickly becoming a mature discipline. To achieve the promise of fully effective software risk management, the software must address several continuing challenges.

- Achieve commitment of all key stakeholders (developers, customers, users, maintainers, and others) to a risk management approach.

● Establish an evolving knowledge base of risk management experience and expertise, organized for easy and collaborative use by all stakeholders.

● Define and propagate mature guidelines on when and how to avoid, prevent, transfer, or accept and manage risk.

● Develop metrics and tools for reasoning about risk management's return on- investment issues, including guidelines for deciding how much of a risk reduction activity is enough. Tools that might be used in this way include risk-focused prototyping, specifying, testing, formal verification and validation, configuration management, and quality assurance.

Guideline Application Tool

1. Description

Risk management template must be published in Twiki and every software project must ensure that risks are evaluated as part of the quality plan (see *4.2.1.25. Quality Plan Document Template*).

7.2.2.3. Customer representative

ISO 90003:2004 Guideline

The customer may have responsibilities under the contract. Particular issues may include the need for the customer to co-operate with the organization, to provide necessary information in a timely manner, and to resolve action items. When assigned to monitor life cycle activities, a customer representative may represent the eventual users of the product, as well as executive management, and have the authority to deal with contractual matters which include, but are not limited to, the following:

a) dealing with customer-supplied software items, data, facilities and tools that are found unsuitable for use;

b) organizing access to end-users, where appropriate.

Review of requirements may be performed by internal or external organizations. This may include reviews of requirements related to contracts, engineering, maintenance or quality.

NOTE For further information on requirements review see ISO/IEC 12207:1995[11], 5.2.1 (supply process — initiation),

5.2.6 (supply process — review and evaluation), 6.4.2.1 (contract verification), and 6.6 (joint review process). For further information on risk management see ISO/IEC 12207:1995/Amd.1:2002[12].

Guideline Application

1. Description

A customer representative is an individual who represents a community that intends to purchase a product. The term is most often applied to a representative of a company who works closely with a producer or developer to clarify specifications for a product or service.

The term is used in software engineering. The customer representative ensures that customer's comments and concerns are channelled through one person. The project team is not bombarded with contradicting messages from various customer departments at times, or made uncomfortable by the customer's lack of response to their queries.

Customer representative must be specified in the *Project Plan* document.

Guideline Application Tool

1. Description

Customer representative must be specified in the contract with the customer, and in the project using *Xuse* and *site* documentation generated by *Maven*.

7.2.3. Customer communication

ISO 9001:2008 Standard

The organization shall determine and implement effective arrangements for communicating with customers in relation to

a) product information,

b) enquiries, contracts or order handling, including amendments, and

c) customer feedback, including customer complaints.

7.2.3.1. General

For computer software, the method of communication may vary depending on the type of contractual agreement, and on the scope of the contract for development, operations or maintenance.

The following guidance for communicating with customers is separated into advice for development and advice for operations/ maintenance life cycle processes.

Guideline Application

1. Description

Customer communications define a convergent set of information technology solutions that together provide software communication professionals the ability to advance the way that they communicate with their customers.

Customer involvement throughout the project life cycle ensures better acceptability of the project. By keeping in touch with the customer all through the project duration, misunderstandings are detected and resolved early. The earlier the problems are detected, the easier they are to solve. By planning and budgeting for adequate customer involvement, the project manager can improve the chances of successful project execution.

Customer involvement could include some or all of the following:

- Keeping the customer informed of the progress of the project
- Making the customer review and approve all documents that represent the work done in one phase and are to be used as basis for subsequent phases,
- Customer may be required to supply some information or support for the project team to perform their work
- Customer may be required to provide test data
- Providing some of the resources (such as hardware, software, documents) may be a customer responsibility and interaction with customer representatives may be required to review project issues and resolve them - typically through a constituted steering committee.

Customer is keep informed of the project progress through periodic reports sent to them. These reports should be reasonably detailed -the format and frequency for these should be mutually agreed with the customer at the start of the project.

An important activity is the approval by the customer (which signed the document) on various intermediate deliverables like the Requirements Specification document. This ensures that the customer is involved in and agrees to the work done so far. Often project managers do not plan for the iterations that may take place while explaining the documents to the customer, providing clarification on it, or amending it to incorporate rate the customer's comments and feedback.

For any stage where someone is studying a document and may have comments/queries, the plan must budget for the interaction and rework that may be required.

The project plan must detail out the extent of customer interaction required, manner and frequently in which this interaction will proceed.

Guideline Application Tool

1. Description

No tool is needed. Customer must sign copy-papers with intermediate deliverables documents (like the *requirements specification document*, *user stories*, *use cases*, and of course *unit acceptance test* results), in periodical meetings.

7.2.3.2. Customer communications during development

ISO 90003:2004 Guideline

Joint reviews involving the organization and the customer may be scheduled on a regular basis, or at significant project events, to cover the following aspects, as appropriate:

a) **product information**, including

1) development plans,

2) conformance of outputs, such as design and development documents, to the customer's agreed

requirements,

3) demonstrations of outputs of the development processes, such as prototypes, and

4) acceptance test results;

b) **enquiries, contracts and amendments**, including

1) the progress of activities concerning the eventual users of the system under development, such as deployment and training,

2) the progress of software development work undertaken by the organization,

3) the progress of agreed activities being undertaken by the customer,

4) the processing of risk management issues, problems and change control items, and

5) the methods by which the customer will be advised of current or planned future changes.

Guideline Application

See section *7.2.3.1 General*.

7.2.3.3. Customer communication during operations and maintenance

ISO 90003:2004 Guideline

Sources of information that involve customer communication in operations and maintenance may include the following:

a) product information, including

1) online help, user manuals describing the product and its use,

2) descriptions of new releases and upgrades, and

3) product web sites;

b) enquiries, contracts and amendments, including

1) progress on product or service delivery, and/or maintenance activities, and

2) processing service or product risks, issues and change requests;

c) customer feedback, including

1) help desk arrangements and effectiveness,

2) progress on customer complaints processing, and

3) surveys, user groups, conferences.

NOTE For further information, see the following:

— ISO/IEC 12207:1995[11], 6.6 (joint reviews process), 5.2.5 (supply process — execution and control), 5.2.6 (supply process — review and evaluation) and 5.2.7 (supply process — delivery and

completion), and ISO/IEC 12207:1995/Amd.1:2002[12], F.1.4.2 (customer support).

— ISO/IEC 14764:1999[19] (software maintenance), 6.8.1 (maintainability and the development process), 7.3.3 (guidelines for a maintenance plan) and 8.2 (problem and modification) to 8.2.3 (controls).

Guideline Application

1. Description

See section *7.2.3.1 General*.

7.3. Design and development

7.3.1. Design and development planning

ISO 90003:2004 Standard

The organization shall plan and control the design and development of product.

During the design and development planning, the organization shall determine

a) the design and development stages,

b) the review, verification and validation that are appropriate to each design and development stage, and

c) the responsibilities and authorities for design and development.

The organization shall manage the interfaces between different groups involved in design and development to ensure effective communication and clear assignment of responsibility.

Planning output shall be updated, as appropriate, as the design and development progresses.

NOTE Design and development review, verification and validation have distinct purposes. They can be conducted and recorded separately or in any combination, as suitable for the product and the organization.

7.3.1.1. Design and development planning

ISO 90003:2004 Guideline

Design and development should be carried out in a disciplined manner to prevent or minimize the occurrence of problems. This approach reduces dependence on verification and validation as the sole methods for identifying problems. The organization should therefore ensure that the software products are developed in compliance with specified requirements and in accordance with design and development planning and/or quality planning (see 7.1 for quality planning).

NOTE 1 ISO/IEC 12207:1995[11] and ISO/IEC 12207:1995/Amd.1:2002[12] include quality planning and development planning as a single planning activity leading to the creation of project management plan(s). A mapping table is provided in Annex B, to show how the items in 7.1.1 and 7.3.1 are satisfied by the related items in ISO/IEC 12207:1995[11], 5.2.4.5, 5.3.1.4 and 6.3.1.3.

NOTE 2 Some items in the following list have been included in the quality planning list in 7.1.2. These are noted in square brackets.

Design and development planning should address the following items, as appropriate:

a) the activities of requirements analysis, design and development, coding, integration, testing, installation and support for acceptance of software products; this includes the identification of, or reference to:

1) activities to be carried out;

2) required inputs to each activity;

3) required outputs from each activity;

4) verification required for each activity output [as 7.1.2 g) – see also 7.3.5];

5) management and supporting activities to be carried out;

6) required team training [as 7.1.2 k)];

b) planning for the control of product and service provision;

c) the organization of the project resources, including the team structure, responsibilities, use of suppliers and material resources to be used;

d) organizational and technical interfaces between different individuals or groups, such as sub-project teams, suppliers, partners, users, customer representatives, quality assurance representative (see 7.3.1.4);

e) the analysis of the possible risks, assumptions, dependencies and problems associated with the design and development;

f) the schedule identifying:

1) the stages of the project [see also 7.1.2 j)];

2) the work breakdown structure;

3) the associated resources and timing;

4) the associated dependencies;

5) the milestones;

6) verification and validation activities [as 7.1.2 g)];

g) the identification of:

1) standards, rules, practices and conventions, methodology, life cycle model, statutory and regulatory requirements [as 7.1.2 d) and e)];

2) tools and techniques for development, including the qualification of, and configuration controls placed on, such tools and techniques;

3) facilities, hardware and software for development;

4) configuration management practices [as 7.1.2 h)];

5) method of controlling nonconforming software products;

6) methods of control for software used to support development;

7) procedures for archiving, back-up, recovery, and controlling access to software products;

8) methods of control for virus protection;

9) security controls;

h) the identification of related planning (including planning of the system) addressing topics such as quality (see 7.1), risk management, configuration management, supplier management, integration, testing (see 7.3.6), release management, installation, training, migration, maintenance, re-use, communication and measurement;

i) documentation control including document/record archive and distribution.

For a COTS product in which the organization does not have control over the design, the organization should assure that the product meets the acceptance criteria.

Planning should be reviewed periodically and any plans amended if appropriate.

NOTE A document defining design and development planning and any of these related planning topics may be an independent document, a part of another document or composed of several documents.

Guideline Application

1. Description

Once the user stories and system requirements are collected, the next step is named the use cases, and create an order for their implementation.

For example, before use case *Customer Login*, is necessary to make the use case of *Customer Registration Account*. Project

manager must establish an order in the implementation of the use cases.

Once use cases are named, ordered, and the traceability from requirements to uses cases is created, an schedule is done, with periods of one week (if the use case has low level difficulty), two weeks (medium difficulty) or three weeks (high difficulty).

The next step is assigning the developers to every use case; they will have the user stories, the requirements, and the name of the use case.

Therefore, the developer must do the following issues, in the next order:

1. Create the use case based on the requirements and all necessary entries in the data dictionary (in *Xuse*).

2. Develop the use case, applying design patterns (if necessary).

Design and development planning is included in section *7.1.1 Software life cycle*.

Guideline Application Tool

1. Description

The main tool for this section is *Microsoft Project* (see *7.1.1 Software life cycle)*.

7.3.1.2. Review, verification and validation

ISO 90003:2004 Guideline

The review, verification and validation for software design and development are covered in 7.3.4 to 7.3.6. In software operations and maintenance, they may be covered in service level agreements or maintenance procedures.

Guideline Application

1. Description

It is not necessary add any guideline. See section *7.3.4* to *7.3.6*.

Guideline Application Tool

1. Description

See section *7.3.4* to *7.3.6*.

7.3.1.3. Responsibilities and authorities

ISO 90003:2004 Guideline

There is no specific guidance.

Guideline Application

1. Description

Responsibilities and authorities are described in section *5.5.1. Responsibility and Authority*.

Guideline Application Tool

1. Description

See section *5.5.1. Responsibility and Authority*.

7.3.1.4. Interfaces

ISO 90003:2004 Guideline

The boundaries of responsibility for each part of the software product and the way that technical information will be transmitted between all parties should be clearly defined in the design and development planning of suppliers. The organization may require review of a supplier's design and development planning.

In defining interfaces, care should be taken to consider parties, other than the customer and organization, who have an interest in the design and development, installation, operation, maintenance and training activities.

These may include customer representatives, suppliers, partners, quality assurance representatives, engineering process group representatives, regulatory authorities, associated development project staff and help desk staff. In particular, the end-users and any intermediate operations function may need to be involved to ensure that appropriate capacity and training are available to achieve committed service levels.

NOTE 1 For further information on design and development planning see ISO/IEC 12207:1995[11], 5.2.4 (planning) and 5.3.1 (development process implementation).

NOTE 2 For further information on software project management see ISO/IEC TR 16326:1999[30], 6.2.2 (planning).

Guideline Application

1. Description

The interface concept is one of the cornerstones of object oriented programming languages. Interfaces differ from classes in that they don't include any member implementation. An interface can have properties and method interfaces only. An interface does not have any method implementation. It can include a method declaration but no function body.

By itself, interfaces are not very useful. You cannot create instances of interfaces. You can implement interfaces by useful classes from which you can create instances. Interfaces are instrumental in enforcing object oriented design methodologies. One group may specify the interface for other groups to follow. Each group may implement its own function bodies, but all groups will adhere to the same interface, defined a priori. It comes back to the first and foremost incentive for object oriented programming: encapsulation. You don't care how a capability is implemented. You, as a programmer, are exposed only to the interface. This is also a good way to watch after the system architecture. You don't let everyone on the team define his or her own interfaces. One central group will usually define them for the rest. Interfaces are a good way to define methods for the whole project.

In Booking Books, I have used interfaces in the main software artefacts, as Managers or DAOs (see *4.2.1.20. Software Micro Architecture*).

Guideline Application Tool

1. Description

The interfaces are programmed in *J2EE*.

7.3.2. Design and development inputs

ISO 9001:2008 Standard

Inputs relating to product requirements shall be determined and records maintained (see 4.2.4). These inputs shall include

a) functional and performance requirements,

b) applicable statutory and regulatory requirements,

c) where applicable, information derived from previous similar designs, and

d) other requirements essential for design and development.

These inputs shall be reviewed for adequacy. Requirements shall be complete, unambiguous and not in conflict with each other.

ISO 90003:2004 Guideline

In system architectural design, system requirements are allocated to hardware, software components and manual operations. The inputs to software requirements analysis are the system requirements allocated to software and specifications of the interfaces between the system components.

For guidance on ISO 9001:2008, 7.3.2 items a), b) and d), see 7.2.1.

Design and development input may be determined from functional, performance, quality, relevant safety and security requirements and system design constraints, or derived through techniques such as prototyping.

Design and development input may also be determined from design change requests originating from previous phases in the iterative development model (cycle), problems to be fixed, or requirements arising from acceptance criteria. Input may also come from contract review activities.

When design and development input documents are being reviewed (this often happens in conjunction with the customer), they should be checked for

a) ambiguities and contradictions,

b) inconsistent, incomplete or unfeasible information or requirements,

c) unrealistic performance specifications,

d) requirements that cannot be verified or validated,

e) unstated or assumed requirements,

f) inaccurate description of user environment and actions,

g) lack of design and development decisions in a requirements document, and

h) omissions of key performance measures.

NOTE For further information see ISO/IEC 9126-1:2001[5] for software quality requirements as software quality characteristics.

Guideline Application

1. Description

The design and development inputs for this project have been the user stories and the use cases.

2. Features

2.1. User Stories

A user story is a software system requirement formulated as one or more sentences in the everyday or business language of the user. User stories are used for the specification of requirements. Each user story is limited, so it fits on a small paper note card to ensure that it does not grow too large. The user stories should be written by the customers for a software project and are their main instrument to influence the development of the software.

User stories are a quick way of handling customer requirements without having to elaborate vast formalized requirement documents and without performing overloaded administrative tasks related to maintaining them. The intention of the user story is to be able to respond faster and with less overhead to rapidly changing real-world requirements.

A user story is an informal statement of the requirement as long as the correspondence of acceptance testing procedures is lacking. Before a user story is to be implemented, an appropriate acceptance procedure must be written by the customer to ensure by testing or otherwise determine whether the goals of the user story have been fulfilled. Some formalization finally happens when the developer accepts the user story and the acceptance procedure as a work specific order.

2.2. Requirements

Requirements analysis encompasses those tasks that go into determining the needs or conditions to meet for a new or altered product, taking account of the possibly conflicting requirements of the various stakeholders, such as beneficiaries or users.

Requirements analysis is critical to the success of a development project. Requirements must be documented, actionable, measurable, testable, related to identified business needs or opportunities, and defined to a level of detail sufficient for system design.

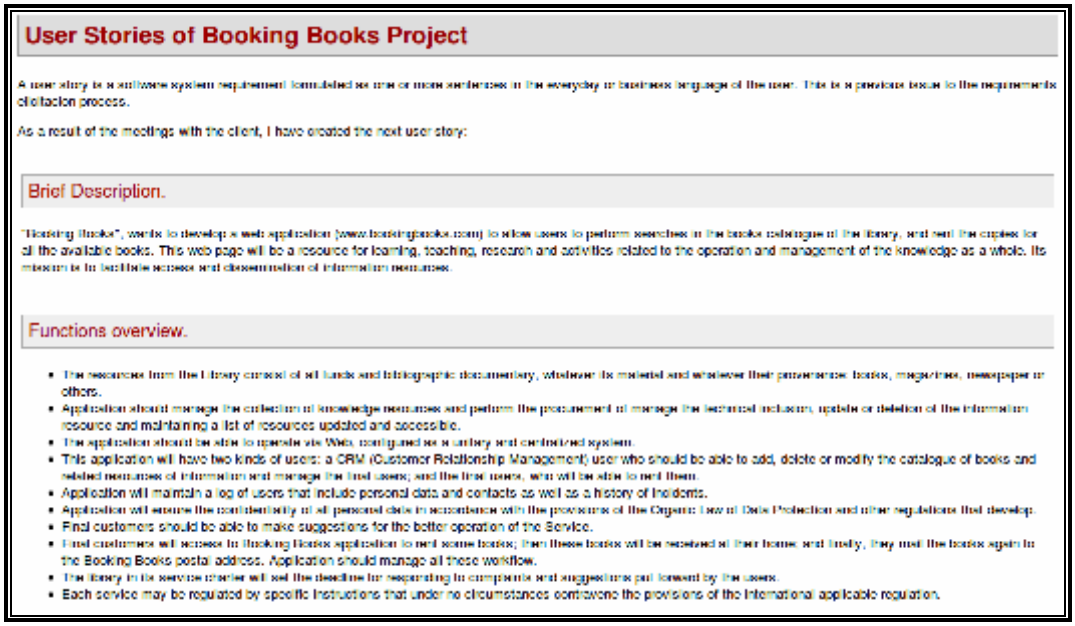
Guideline Application Tool

1. Description

I have done the user stories and requirements using Xuse.

2. Features

2.1. User Stories



Picture III-43: User Stories of Booking Books Project

2.2. Requirements

	id	v.	intent	title	description
⊕	RQ001	1.0.0	shall	**Customer account registration**	Customer shall register a new account in the system in order to rent **bibliographic found**s.
⊕	RQ002	1.0.0	shall	**System email notification**	System shall notify the customer's accounts creation by an email.
⊕	RQ003	1.0.0	shall	**Customer account username (login)**	The username (login) of the customer shall be unique in all the system.
⊕	RQ004	1.0.0	shall	**Customer account email**	The mail address of the customer shall be unique in all the system.
⊕	RQ005	1.0.0	shall	**Remember password process**	When an customer doesn't remember his password, system will create and send a new one to the customer's email address, using his username.
⊕	BR001	1.0.0	shall	**Password regeneration process**	When an Customer requests for a new password, it must be a password with 10 characters and a combination of letters and numbers: [a-z][A-Z][0-9] generated ramdonly.
⊕	BR002	1.0.0	shall	**Search process.**	When actor performs a search, at least one search criteria must be added to the form.
⊕	RQ009	1.0.0	shall	**Search operations**	Customers shall perform searches over the **bibliographic found**s of the library by name, publish date and by author.
⊕	RQ010	1.0.0	shall	**Bibliographic founds types**	Bibliographic founds shall be: books, magazines and newspapers.

Picture III-44: Functional Requirements Diagram (I)

⊕	RQ011	1.0.0	shall	**Rent operations**	In order to rent some **bibliographic found**, the customer shall be logged in.
⊕	RQ012	1.0.0	shall	**Bibliographic founds management**	CRM user shall manage (inclusion, update or deletion) all the **bibliographic found**s.
⊕	RQ014	1.0.0	shall	**Booking Books user types**	System will have two kinds of users: CRM (Customer Relationship Management) user and customers.
⊕	RQ015	1.0.0	shall	**CRM users description**	CRM users shall perform **CRUD** operations over the **bibliographic found**s and over the customers.
⊕	RQ018	1.0.0	shall	**Customers should be able to make suggestions for the better operation of the Service.**	System must provide an email account, in order to have customers suggestions about the service.
⊕	RQ019	1.0.0	shall	**CRM users shall manage all rent workflow**	CRM users shall manage all the rent workflow: **bibliographic found**s will be rented by customers and received at their postal address; finally, they will send the books again to the Booking Books postal address.
⊕	RQ024	1.0.0	shall	**System email notification**	System must be able to notify any information to the users thru an email.
⊕	RQ025	1.0.0	shall	**General policies management**	The CRM user must be able to edit the general policies for Booking Books, and for final clients. This policies will be shown in different forms and pages along the web site.
⊕	RQ026	1.0.0	shall	**Delete users**	When a customer is deleted, the system must show an alert to confirm this action.

Picture III-45: Functional Requirements Diagram (II)

	id	v.	intent	title	description
	RQ006	1.0	shall	**Remember Password with unregistered username.**	When an Customer requests for a new password with a unregistered username, for security reasons system doesn't inform to the actor about the unregistered username.
	RQ007	1.0	shall	**Username of a customer can't be modified**	When an Customer modifies his account, he shall modify all his data but the username.
	RQ008	1.0	shall	**Register operations**	System shall register some operations when it happens in the system. • log in • log out • rent of a Bibliographic found • devolution of a Bibliographic found • **bibliographic found** created • **bibliographic found** deleted
	RQ020	1.0	shall	**Communication over Internet shall be encripted**	Communication over Internet shall be encripted for security reasons.
	RQ021	1.0	shall	**Access to CRM pages must be authenticated**	Access to CRM pages must be authenticated.
	RQ022	1.0	shall	**Access to customer pages must be authenticated**	Access to customer pages must be authenticated.

Picture III-46: Security Requirements Diagram

	id	v.	intent	title	description
	RQ013	1.0	shall	**Application shall be accesible by Web, as an unitary and centralized system**	Users shall access application by Web, configured as a unitary and centralized system.

Picture III-47: Architectural Requirements

	id	v.	intent	title	description
	RQ016	1.0	shall	**System logs operations**	System will maintain a log of users that include personal data and contacts as well as a history of incidents.
	RQ017	1.0	shall	**Protection Data Clause**	Application will ensure the confidentiality of all personal data in accordance with the provisions of the Organic Law of Data Protection and other regulations that develop.

Picture III-48: Legal Requirements

	id	v.	intent	title	description
	RQ027	1.0	shall	**Supported web page navigators**	Application must work fine with firefox, explorer.

Picture III-49: Operational Requirements

7.3.3. Design and development outputs

ISO 9001:2008 Standard

The outputs of design and development shall be in a form suitable for verification against the design and development input and shall be approved prior to release.

Design and development outputs shall

a) meet the input requirements for design and development,

b) provide appropriate information for purchasing, production and for service provision,

c) contain or reference product acceptance criteria, and

d) specify the characteristics of the product that are essential for its safe and proper use.

NOTE Information for production and service provision can include details for the preservation of product.

ISO 90003:2004 Guideline

The output from the design and development process should be defined and documented in accordance with the prescribed or chosen method. This output should be complete, accurate and consistent with the requirements, and may be produced using computer design and development tools. Design and development outputs may be expressed in textual form, by diagrams or using symbolic modelling notation, and may include

a) design, development and test specifications,

b) data models,

c) pseudo code or source code,

d) user guides, operator documentation, training material, maintenance documentation,

e) developed product, and

f) formal methods.

Prototyping, when used, should result in design and development (output) documentation.

The acceptance criteria for design and development outputs should be defined in order to demonstrate that the inputs to each design and development stage are correctly reflected in the outputs.

Tools should be validated for their specific intended use (see 7.3.6 and 7.6).

Guideline Application

1. Description

For the design and development outputs, I have created use cases, core reports and I have added the source code of the most important classes.

2. Features

2.1. Use Cases

A **use case** is a description of a system's behaviour as it responds to a request that originates from outside of that system. In other words, a use case describes "who" can do "what" with the system in question. The use case technique is used to capture a system's behavioural requirements by detailing scenario-driven threads through the functional requirements.

2.2. Forms

Form is called the template or page with empty spaces, which must be filled with a purpose; for example a registration form in which you fill in the gaps with the required personal information.

2.3. Core Reports

These are the reports which include the Model of the Model-View-Controller architecture (Entities, DAOs and Managers).

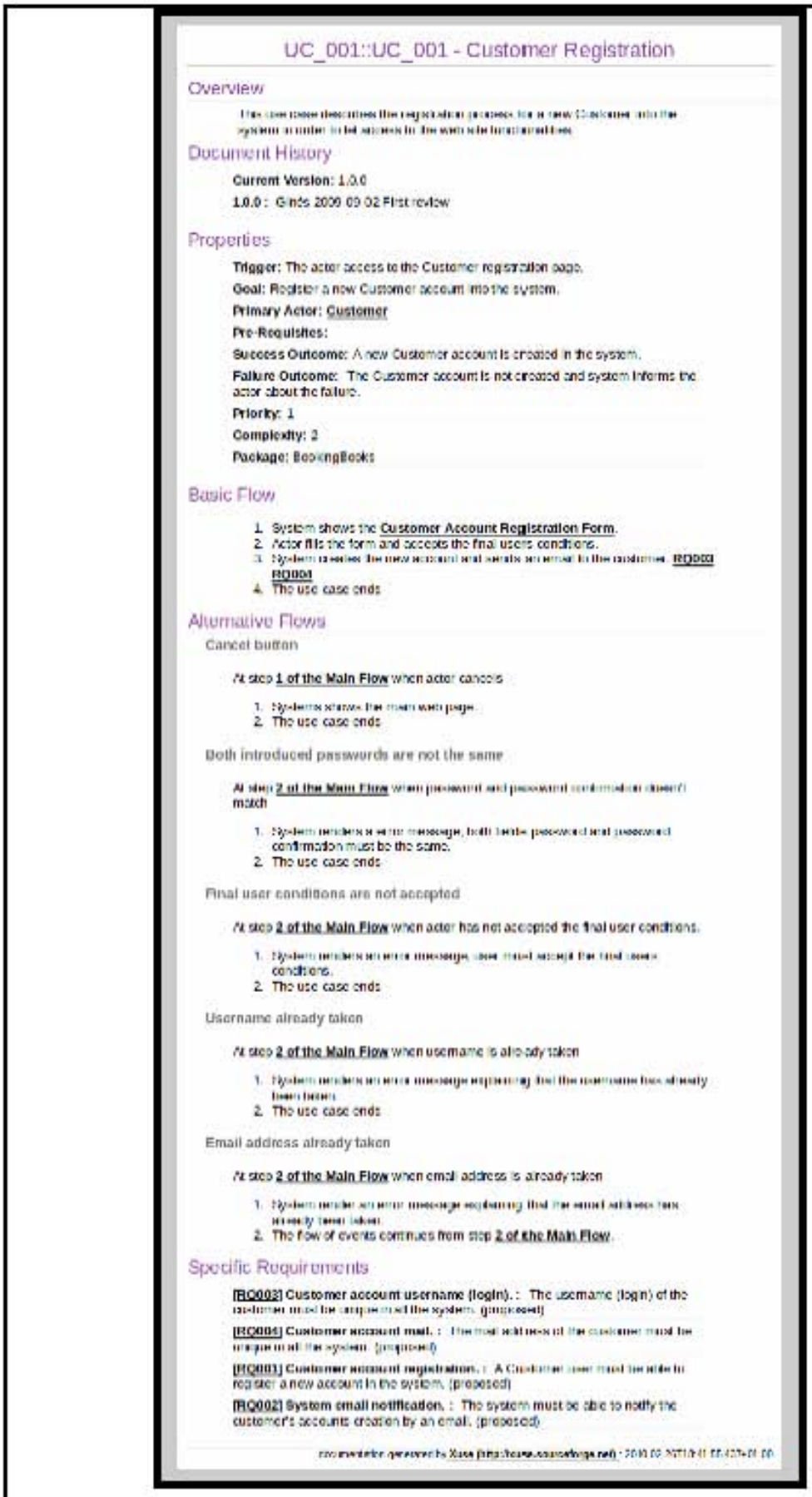
Guideline Application Tool

1. Description

For use cases and core reports I have use Xuse. For source code I haven't use any tool.

2. Features

2.1. Use Cases



UC_001::UC_001 - Customer Registration

Overview

The use case describes the registration process for a new Customer into the system in order to be access to the web site functionalities.

Document History

Current Version: 1.0.0

1.0.0 : Gmds 2009 03 02 First review

Properties

Trigger: The actor access to the Customer registration page.

Goal: Register a new Customer account into the system.

Primary Actor: Customer

Pre-Requisites:

Success Outcome: A new Customer account is created in the system.

Failure Outcome: The Customer account is not created and system informs the actor about the failure.

Priority: 1

Complexity: 2

Package: BookingBooks

Basic Flow

1. System shows the Customer Account Registration Form.
2. Actor fills the form and accepts the final users conditions.
3. System creates the new account and sends an email to the customer. RQ003 RQ004
4. The use case ends

Alternative Flows

Cancel button

At step 1 of the Main Flow when actor cancels

1. System shows the main web page.
2. The use case ends

Both introduced passwords are not the same

At step 2 of the Main Flow when password and password confirmation doesn't match

1. System renders a error message, both fields password and password confirmation must be the same.
2. The use case ends

Final user conditions are not accepted

At step 2 of the Main Flow when actor has not accepted the final user conditions.

1. System renders an error message, user must accept the final users conditions.
2. The use case ends

Username already taken

At step 2 of the Main Flow when username is already taken

1. System renders an error message explaining that the username has already been taken.
2. The use case ends

Email address already taken

At step 2 of the Main Flow when email address is already taken

1. System renders an error message explaining that the email address has already been taken.
2. The flow of events continues from step 2 of the Main Flow.

Specific Requirements

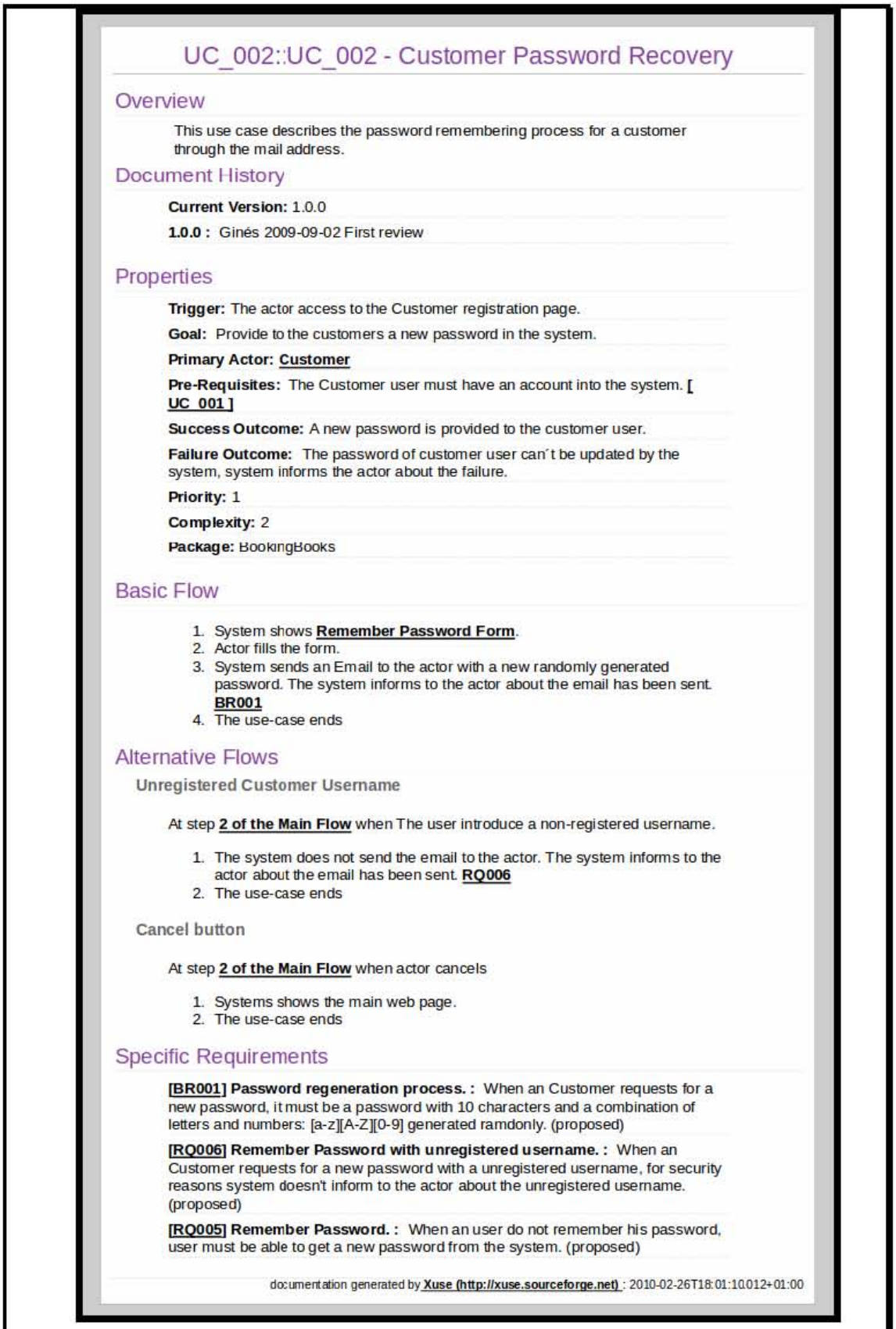
[RQ003] Customer account username (login). : The username (login) of the customer must be unique in the system. (proposed)

[RQ004] Customer account mail. : The mail address of the customer must be unique in the system. (proposed)

[RQ001] Customer account registration. : A Customer user must be able to register a new account in the system. (proposed)

[RQ002] System email notification. : The system must be able to notify the customer's accounts creation by an email. (proposed)

Picture III-50: Use Case 001: Customer Registration



UC_002::UC_002 - Customer Password Recovery

Overview

This use case describes the password remembering process for a customer through the mail address.

Document History

Current Version: 1.0.0

1.0.0 : Ginés 2009-09-02 First review

Properties

Trigger: The actor access to the Customer registration page.

Goal: Provide to the customers a new password in the system.

Primary Actor: **Customer**

Pre-Requisites: The Customer user must have an account into the system. **[UC_001]**

Success Outcome: A new password is provided to the customer user.

Failure Outcome: The password of customer user can´t be updated by the system, system informs the actor about the failure.

Priority: 1

Complexity: 2

Package: BookingBooks

Basic Flow

1. System shows **Remember Password Form**.
2. Actor fills the form.
3. System sends an Email to the actor with a new randomly generated password. The system informs to the actor about the email has been sent. **BR001**
4. The use-case ends

Alternative Flows

Unregistered Customer Username

At step **2 of the Main Flow** when The user introduce a non-registered username.

1. The system does not send the email to the actor. The system informs to the actor about the email has been sent. **RQ006**
2. The use-case ends

Cancel button

At step **2 of the Main Flow** when actor cancels

1. Systems shows the main web page.
2. The use-case ends

Specific Requirements

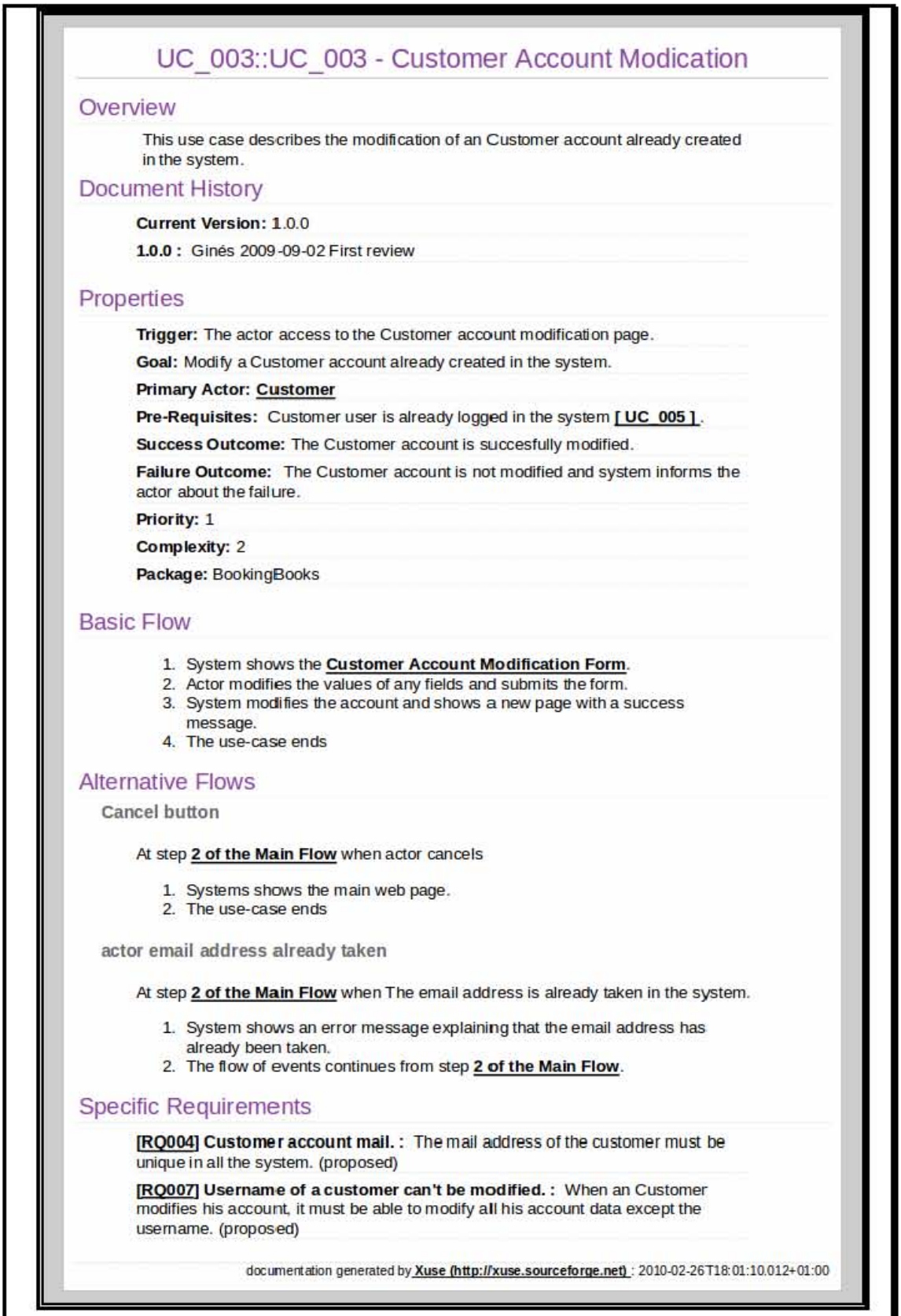
[BR001] Password regeneration process. : When an Customer requests for a new password, it must be a password with 10 characters and a combination of letters and numbers: [a-z][A-Z][0-9] generated ramdonly. (proposed)

[RQ006] Remember Password with unregistered username. : When an Customer requests for a new password with a unregistered username, for security reasons system doesn't inform to the actor about the unregistered username. (proposed)

[RQ005] Remember Password. : When an user do not remember his password, user must be able to get a new password from the system. (proposed)

documentation generated by **Xuse (http://xuse.sourceforge.net)** : 2010-02-26T18:01:10.012+01:00

Picture III-51: Use Case 001: Customer Password Recovery



UC_003::UC_003 - Customer Account Modication

Overview

This use case describes the modification of an Customer account already created in the system.

Document History

Current Version: 1.0.0

1.0.0 : Ginés 2009-09-02 First review

Properties

Trigger: The actor access to the Customer account modification page.

Goal: Modify a Customer account already created in the system.

Primary Actor: **Customer**

Pre-Requisites: Customer user is already logged in the system **[UC_005]** .

Success Outcome: The Customer account is succesfully modified.

Failure Outcome: The Customer account is not modified and system informs the actor about the failure.

Priority: 1

Complexity: 2

Package: BookingBooks

Basic Flow

1. System shows the **Customer Account Modification Form**.
2. Actor modifies the values of any fields and submits the form.
3. System modifies the account and shows a new page with a success message.
4. The use-case ends

Alternative Flows

Cancel button

At step **2 of the Main Flow** when actor cancels

1. Systems shows the main web page.
2. The use-case ends

actor email address already taken

At step **2 of the Main Flow** when The email address is already taken in the system.

1. System shows an error message explaining that the email address has already been taken.
2. The flow of events continues from step **2 of the Main Flow**.

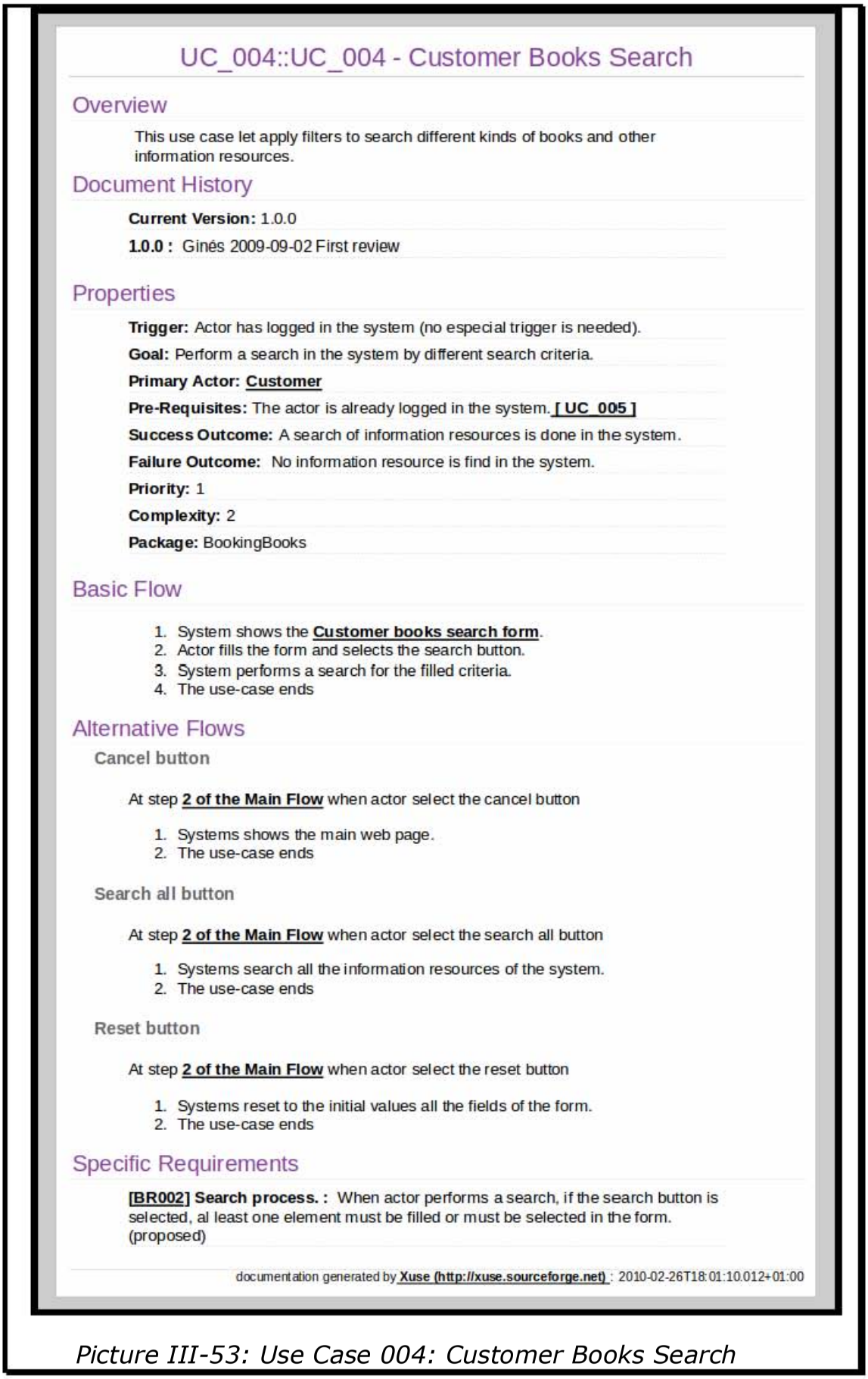
Specific Requirements

[RQ004] Customer account mail. : The mail address of the customer must be unique in all the system. (proposed)

[RQ007] Username of a customer can't be modified. : When an Customer modifies his account, it must be able to modify all his account data except the username. (proposed)

documentation generated by **Xuse (http://xuse.sourceforge.net)** : 2010-02-26T18:01:10.012+01:00

Picture III-52: Use Case 003: Customer Account Modification



UC_004::UC_004 - Customer Books Search

Overview

This use case let apply filters to search different kinds of books and other information resources.

Document History

Current Version: 1.0.0

1.0.0 : Ginés 2009-09-02 First review

Properties

Trigger: Actor has logged in the system (no especial trigger is needed).

Goal: Perform a search in the system by different search criteria.

Primary Actor: **Customer**

Pre-Requisites: The actor is already logged in the system. **[UC_005]**

Success Outcome: A search of information resources is done in the system.

Failure Outcome: No information resource is find in the system.

Priority: 1

Complexity: 2

Package: BookingBooks

Basic Flow

1. System shows the **Customer books search form**.
2. Actor fills the form and selects the search button.
3. System performs a search for the filled criteria.
4. The use-case ends

Alternative Flows

Cancel button

At step **2 of the Main Flow** when actor select the cancel button

1. Systems shows the main web page.
2. The use-case ends

Search all button

At step **2 of the Main Flow** when actor select the search all button

1. Systems search all the information resources of the system.
2. The use-case ends

Reset button

At step **2 of the Main Flow** when actor select the reset button

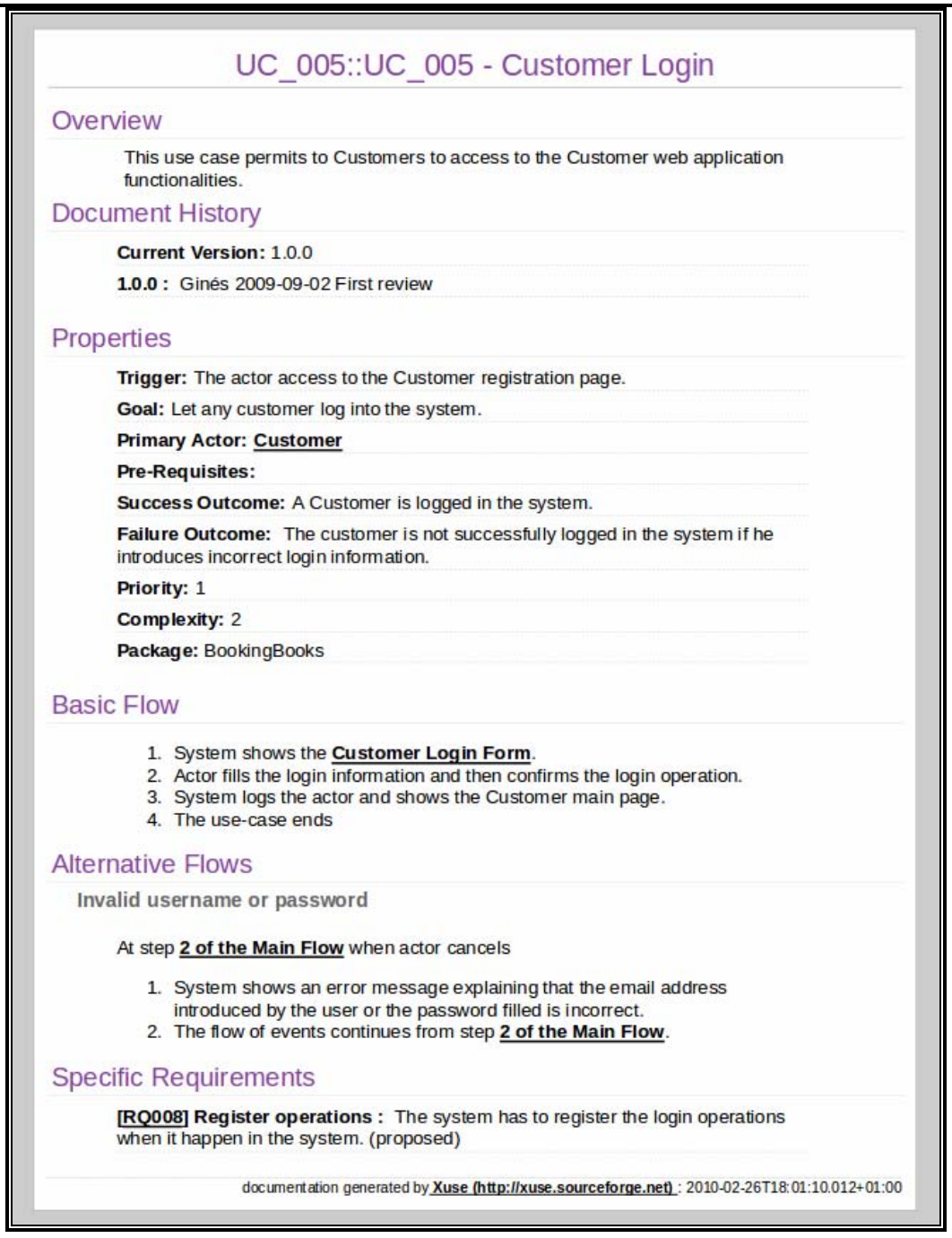
1. Systems reset to the initial values all the fields of the form.
2. The use-case ends

Specific Requirements

[BR002] Search process. : When actor performs a search, if the search button is selected, al least one element must be filled or must be selected in the form. (proposed)

documentation generated by **Xuse (http://xuse.sourceforge.net)** : 2010-02-26T18:01:10.012+01:00

Picture III-53: Use Case 004: Customer Books Search



UC_005::UC_005 - Customer Login

Overview

This use case permits to Customers to access to the Customer web application functionalities.

Document History

Current Version: 1.0.0

1.0.0 : Ginés 2009-09-02 First review

Properties

Trigger: The actor access to the Customer registration page.

Goal: Let any customer log into the system.

Primary Actor: **Customer**

Pre-Requisites:

Success Outcome: A Customer is logged in the system.

Failure Outcome: The customer is not successfully logged in the system if he introduces incorrect login information.

Priority: 1

Complexity: 2

Package: BookingBooks

Basic Flow

1. System shows the **Customer Login Form**.
2. Actor fills the login information and then confirms the login operation.
3. System logs the actor and shows the Customer main page.
4. The use-case ends

Alternative Flows

Invalid username or password

At step **2 of the Main Flow** when actor cancels

1. System shows an error message explaining that the email address introduced by the user or the password filled is incorrect.
2. The flow of events continues from step **2 of the Main Flow**.

Specific Requirements

[RQ008] Register operations : The system has to register the login operations when it happen in the system. (proposed)

documentation generated by **Xuse (http://xuse.sourceforge.net)** : 2010-02-26T18:01:10.012+01:00

Picture III-54: Use Case 005: Customer Login

2.2. Forms

b

bibliographic found Information resource, which can be rented by any customer

Picture III-55: Bibliographic Found

C

CRUD Operations to manage a given entity, namely: Create, Read, Update and Delete

Customer Account Modification Form

Field	Type	Description	Mandatory
Title of the form	Text non editable.	The literal "Customer account modification form"[[customer.modification.form.tile]]	-
Description of the form	Text non editable.	The literal "Through this form you can update your account data." [[customer.modification.form.description]]	-
Access data title	Text non editable.	The literal "Access Data" [[customer.modification.form.access.data]]	-
Password	Text non editable.	The literal "Password" [[customer.modification.form.password]]	-
Password	PasswordField, String[5 , 20], [no restrictions]	The password of the user	yes
Password Confirmation	Text non editable.	The literal "Password confirmation" [[customer.modification.form.password.confirmation]]	-
Password Confirmation	PasswordField, String[5 , 20], [no restrictions]	The password confirmation of the user	yes
Personal data	Text non editable.	The literal "Personal Data" [[customer.modification.form.personal.data]]	-
Name	Text non editable.	The literal "Name of the user" [[customer.modification.form.name]]	-
Name	TextField, String[4 , 50], [a-z][A-Z][0-9] [ñÑçÇáéíóú]	The name of the user	yes
First surname	Text non editable.	The literal "First surname" [[customer.modification.form.first.surname]]	-
First surname	TextField, String[4 , 50], [a-z][A-Z][0-9] [ñÑçÇáéíóú]	The fisrt surname of the user	yes

Picture III-56: Form: Customer Account Modification Form(I)

Second surname	Text non editable.	The literal "Second surname" [[customer.modification.form.second.surname]]	-
Second surname	TextField, String[4 , 50], [a-z][A-Z][0-9] [ñÑçÇáéíóú]	The second surname of the user	yes
Email	Text non editable.	The literal "Email" [[customer.modification.form.email]]	-
Email	TextField, String[7 , 40]. Email type.	The email of the user	yes
Address Data	Text non editable.	The literal "Address data" [[customer.modification.form.address.data]]	-
Post Code	Text non editable.	The litaral "Post Code" [[customer.modification.form.post.code]]	-
Post Code	TextField, String[0 , 10], [0-9]	The post code of the user	no
Country	Text non editable.	The literal "Country" [[customer.modification.form.country]]	-
Country	Select	The country of the user	yes
Province	Text non editable.	The literal "Province" [[customer.modification.form.province]]	-
Province	Select	The province of the user	yes
Locality	Text non editable.	The literal "Locality" [[customer.modification.form.locality]]	-
Locality	Select	The locality of the user	yes
Telephone number	Text non editable.	The literal "Telephone number" [[customer.modification.form.telephcne.number]]	-
Telephone number	TextField, String[6 , 15], [0-9]	The telephone number of the user	yes
Fax number	Text non editable.	The literal "Fax number" [[customer.modification.form.fax.number]]	-
Fax number	TextField, String[0 , 15], [0-9]	The fax number of the user	no
Update account data	Button, meaning update the account data	The literal "Update account data" [[customer.modification.form.update.account.button.text]]	-
Cancel	Button, meaning cancel the operation	The literal "Cancel" [[customer.modification.form.cancel.update.account.button.text]]-	-

Picture III-57: Customer Account Modification Form (II)

Customer Account Registration Form

Field	Type	Description	Mandatory
Title of the form	Text non editable.	The literal "Customer registration form"[[customer.registration.form.tile]]	-
Description of the form	Text non editable.	The literal "Through this form you can register as a customer and thus enjoy all the benefits of Booking Books." [[customer.registration.form.description]]	-
Access data title	Text non editable.	The literal "Access Data" [[customer.registration.form.access.data]]	-
Username	Text non editable.	The literal "Username" [[customer.registration.form.username]]	-
Username	TextField, String[3 , 20], [a-z][A-Z][0-9]	The name of the user	yes
Password	Text non editable.	The literal "Password" [[customer.registration.form.password]]	-
Password	PasswordField, String[5 , 20], [no restrictions]	The password of the user	yes
Password Confirmation	Text non editable.	The literal "Password confirmation" [[customer.registration.form.password.confirmation]]	-
Password Confirmation	PasswordField, String[5 , 20], [no restrictions]	The password confirmation of the user	yes
Personal data	Text non editable.	The literal "Personal Data" [[customer.registration.form.personal.data]]	-
Name	Text non editable.	The literal "Name of the user" [[customer.registration.form.name]]	-
Name	TextField, String[4 , 50], [a-z][A-Z][0-9] [ñÑçÇáéíóú]	The name of the user	yes
First surname	Text non editable.	The literal "First surname" [[customer.registration.form.first.surname]]	-
First surname	TextField, String[4 , 50], [a-z][A-Z][0-9] [ñÑçÇáéíóú]	The fisrt surname of the user	yes
Second surname	Text non editable.	The literal "Second surname" [[customer.registration.form.second.surname]]	-
Second surname	TextField, String[4 , 50], [a-z][A-Z][0-9] [ñÑçÇáéíóú]	The second surname of the user	yes

Picture III-58:Customer Account Registration Form (I)

Address Data	Text non editable.	The literal "Address data" [[customer registration form address data]]	-
Country	Text non editable.	The literal "Country" [[customer registration form country]]	-
Country	Select	The country of the user	yes
Province	Text non editable.	The literal "Province" [[customer registration form province]]	
Province	Select	The province of the user	yes
Locality	Text non editable.	The literal "Locality" [[customer registration form locality]]	
Locality	Select	The locality of the user	yes
Address Data	Text non editable.	The literal "Other access data" [[customer registration form other address data]]	-
Post Code	Text non editable.	The literal "Post Code" [[customer registration form post code]]	-
Post Code	Text field, String[0...10] [0..1]	The post code of the user	no
Email	Text non editable.	The literal "Email" [[customer registration form email]]	-
Email	Text field, String[5...40] Email type.	The email of the user	yes
Telephone number	Text non editable.	The literal "Telephone number" [[customer registration form telephone number]]	-
Telephone number	Text field, String[5...15] [0..1]	The telephone of the user	yes
Fax number	Text non editable.	The literal "Fax number" [[customer registration form fax number]]	-
Fax number	Text field, String[5...15] [0..1]	The fax number of the user	no
Booking Device Access Restrictions	Text non editable.	The literal "Booking Device Access Restrictions" [[customer registration form access restrictions]]	
Protection Data Clause	Button, meaning generate a PDF document with the Protection Data Clause Terms.	The literal "Protection data clause" [[download data conditions terms]]	
Protection Data Clause	Button, meaning generate a PDF document with the Usage Conditions Terms.	The literal "Usage Conditions Terms" [[download terms and conditions use]]	
Accept all conditions terms	Text non editable.	The literal "Accept all conditions terms" [[customer registration form data accept all conditions terms]]	-
Usage conditions terms	Checkbox	Checkbox that must be selected to register the customer account	yes
Create new account	Button, meaning create new account	The literal "Create new account" [[customer registration form create new account button text]]	-
Cancel	Button, meaning cancel the operation	The literal "Cancel" [[customer registration form cancel create new account button text]]	

Picture III-59: Customer Account Registration Form (II)

Customer books search form

Field	Type	Description	Mandatory
Title of the form	Text non editable.	The literal "Customer books search form"[[customer.books.search.form.tile]]	-
Description of the form	Text non editable.	The literal "Through this form you can perform a search of books, magazines and other related information resources." [[customer.books.search.form.description]]	-
Access data title	Text non editable.	The literal "Search criteria" [[customer.books.search.form.search.criteria]]	-
Author	Text non editable.	The literal "Author" [[customer.books.search.form.author]]	-
Author	Text Field, String[3 , 50], [no restrictions]	The author of the information item	no
Publisher	Text non editable.	The literal "Publisher" [[customer.books.search.form.publisher]]	-
Publisher	Text Field, String[3 , 50], [no restrictions]	The publisher of the information item	no
Subject	Text non editable.	The literal "Subject" [[customer.books.search.form.subject]]	-
Subject	TextField, String[3 , 50], [no restrictions]	The subjetc of the information item	no
ISBN	Text non editable.	The literal "ISBN" [[customer.books.search.form.isbn]]	-
ISBN	TextField, String[3 , 50], [0-9]	The ISBN of the information item	no
Publication dates	Text non editable.	The literal "Publication Dates (for example, 1990-1992)" [[customer.books.search.form.publication.dates]]	-
Publication dates	TextField, String[9], [no restrictions]	The publication dates, (two years separated by "-", for example 1990-1992)	no

Picture III-60: Cutomer Books Search Form (I)

Resource type	Text non editable.	The literal "Resource type" [[customer.books.search.form.resource.type]]	-
Resource type	Select Field, with the values "all", "books", "magazines" or "newspapaers".	The resource type	-
Search	Button, meaning to perform a search	The literal "Search" [[customer.books.search.form.search.button.text]]	-
Search all	Button, meaning to perform a search of all the resources of the system	The literal "Search all" [[customer.books.search.form.search.all.button.text]]	-
Reset	Button, meaning reset all the fields to their initial values	The literal "Reset" [[customer.books.search.form.reset.button.text]]-	-
Cancel	Button, meaning cancel the operation	The literal "Cancel" [[customer.books.search.form.cancel.button.text]]-	-

Picture III-61: Customer Books Search Form (II)

Customer Login Form

Field	Type	Description	Mandatory
Title of the form	Text non editable.	The literal "Customer login form"[[customer.login.form.tile]]	-
Username	Text non editable.	The literal "Username" [[customer.login.form.username]]	-
Username	Text Field, String[3 , 20], [no restrictions]	The username	yes
Password	Text non editable.	The literal "Password" [[customer.login.form.password]]	-
Password	Password Field, String[5 , 20], [no restrictions]	The password	yes
Login	Button, meaning log in	The literal "Login" [[customer.login.form.login.button.text]]-	-

Picture III-62:Customer Login Form (I)

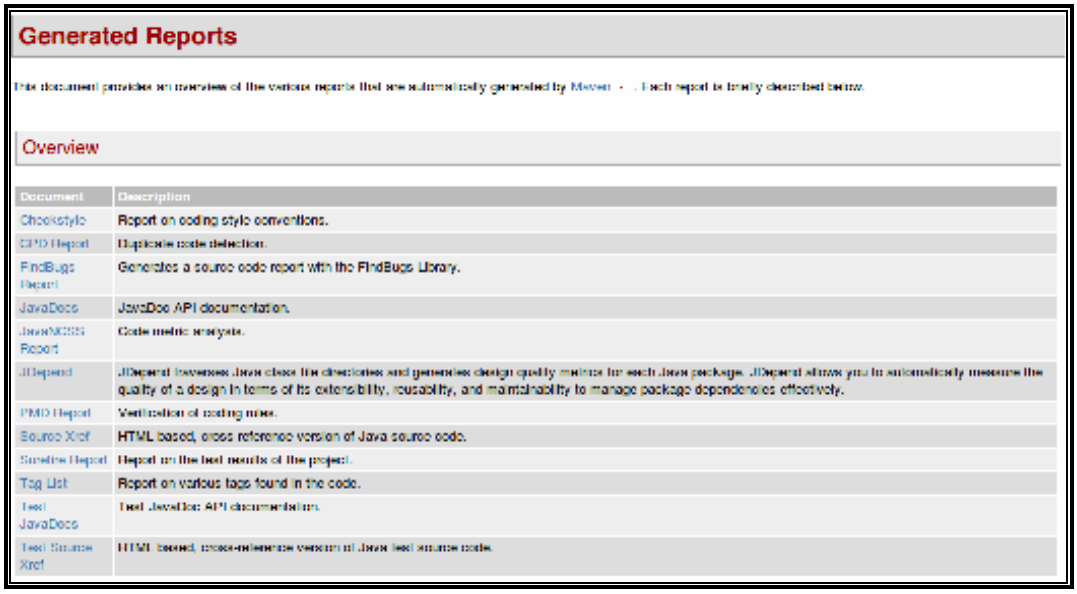
R

Remember Password Form

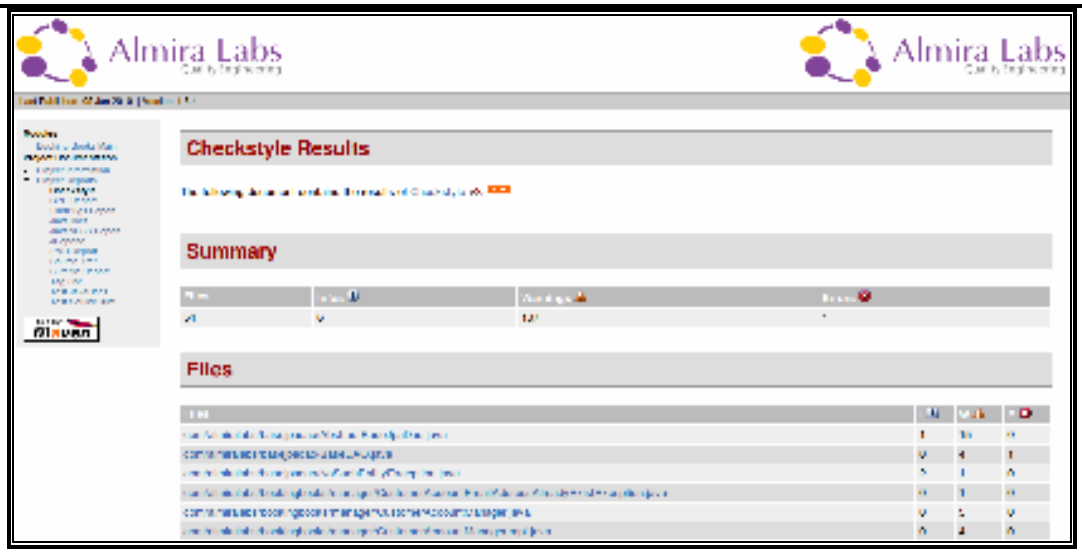
Field	Type	Description	Mandatory
Title of the form	Text non editable.	The literal "Remember password form"[[customer.remember.password.form.tile]]	-
Description of the form	Text non editable.	The literal "Through this form you can send a new password to your email account." [[customer.remember.password.form.description]]	-
Introduce your username	Text non editable.	The literal "Introduce your username" [[customer.remember.password.username]]	-
Username	TextField, String[3 , 20], [a-z][A-Z][0-9]	The name of the user	yes
Send new password	Button, meaning send new password	The literal "Send new password" [[customer.remember.password.form.send.button.text]]	-
Cancel	Button, meaning cancel the operation	The literal "Cancel" [[customer.remember.password.form.cancel.button.text]]-	-

Picture III-63: Remember Password Form

2.3. Core Reports

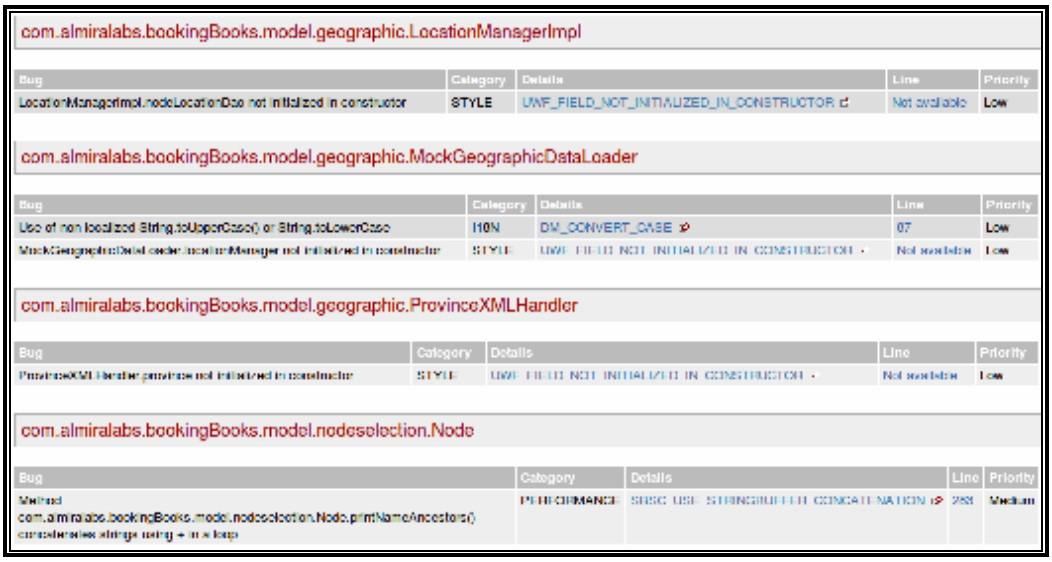


Picture III-64: Generated Reports

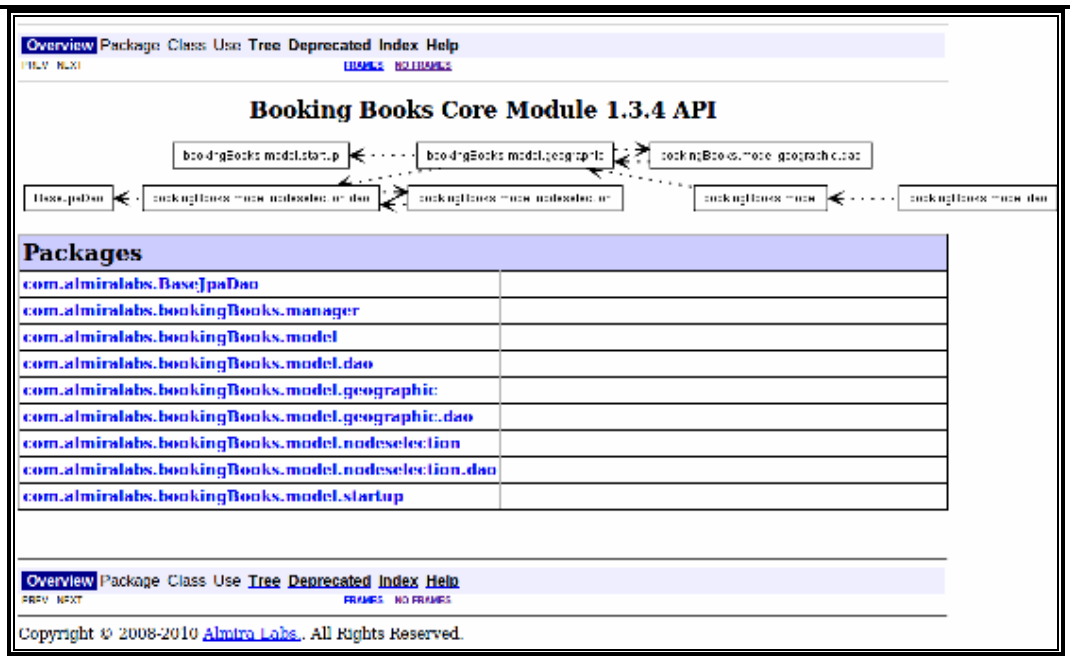


Picture III-65: Checkstyle Report

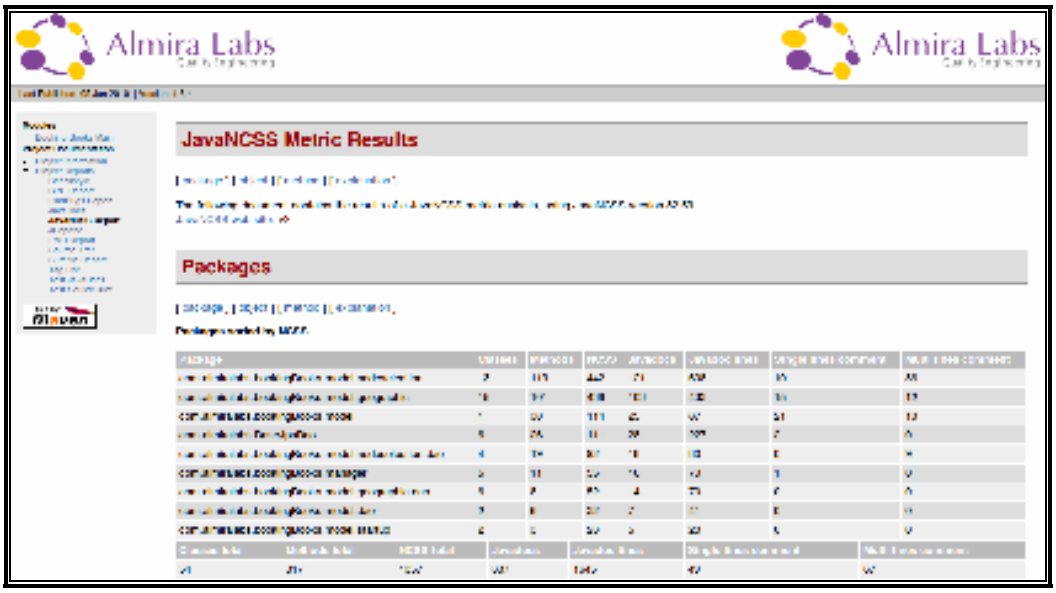
Picture III-66: Find Bugs Detector Report (I)



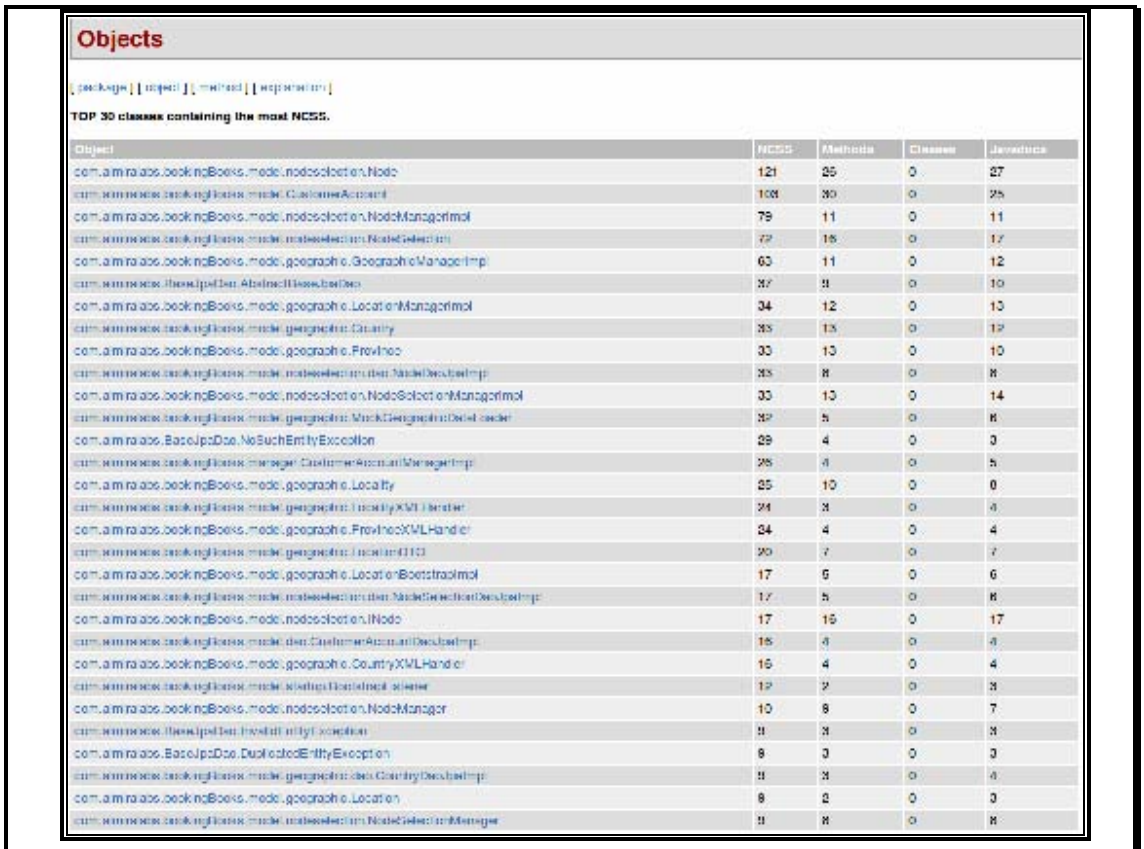
Picture III-67: Find Bugs Detector Report (II)



Picture III-68: Javadoc Report



Picture III-69: JavaNCSS Metric Results Report (I)



Objects

[package] [object] [method] [explanation]

TOP 30 classes containing the most NCSS.

Object	NCSS	Methods	Classes	Javadocs
com.amrraabe.bookingBooks.model.nodeselection.Node	121	35	0	27
com.amrraabe.bookingBooks.model.CustomerAccount	109	30	0	25
com.amrraabe.bookingBooks.model.nodeselection.NodeManagerImpl	79	11	0	11
com.amrraabe.bookingBooks.model.nodeselection.NodeSelection	72	16	0	17
com.amrraabe.bookingBooks.model.geographic.GeographicManagerImpl	63	11	0	12
com.amrraabe.BaseJpaDao.AbstractBaseDao	37	9	0	10
com.amrraabe.bookingBooks.model.geographic.LocationManagerImpl	34	12	0	13
com.amrraabe.bookingBooks.model.geographic.Country	33	13	0	12
com.amrraabe.bookingBooks.model.geographic.Province	33	13	0	10
com.amrraabe.bookingBooks.model.nodeselection.dao.NodeDaoImpl	33	8	0	8
com.amrraabe.bookingBooks.model.nodeselection.NodeSelectionManagerImpl	33	13	0	14
com.amrraabe.bookingBooks.model.geographic.WorldGeographicDataReader	32	5	0	6
com.amrraabe.BaseJpaDao.NoSuchEntityException	29	4	0	3
com.amrraabe.bookingBooks.manager.CustomerAccountManagerImpl	26	4	0	5
com.amrraabe.bookingBooks.model.geographic.Locality	25	10	0	9
com.amrraabe.bookingBooks.model.geographic.LocalityXMLHandler	24	3	0	4
com.amrraabe.bookingBooks.model.geographic.ProvinceXMLHandler	24	4	0	4
com.amrraabe.bookingBooks.model.geographic.LocationUtil	20	7	0	7
com.amrraabe.bookingBooks.model.geographic.LocationBootstrapImpl	17	5	0	6
com.amrraabe.bookingBooks.model.nodeselection.dao.NodeSelectionDaoImpl	17	5	0	6
com.amrraabe.bookingBooks.model.nodeselection.INode	17	16	0	17
com.amrraabe.bookingBooks.model.dao.CustomerAccountDaoImpl	16	4	0	4
com.amrraabe.bookingBooks.model.geographic.CountryXMLHandler	16	4	0	4
com.amrraabe.bookingBooks.model.xstream.XstreamImpl.clone	12	2	0	3
com.amrraabe.bookingBooks.model.nodeselection.NodeManager	10	8	0	7
com.amrraabe.BaseJpaDao.InvalidEntityException	9	3	0	3
com.amrraabe.BaseJpaDao.DuplicatedEntityException	9	3	0	3
com.amrraabe.bookingBooks.model.geographic.dao.CountryDaoImpl	9	3	0	4
com.amrraabe.bookingBooks.model.geographic.Location	9	2	0	3
com.amrraabe.bookingBooks.model.nodeselection.NodeSelectionManager	9	8	0	8

Picture III-70: JavaNCSS Metric Results Report (II)

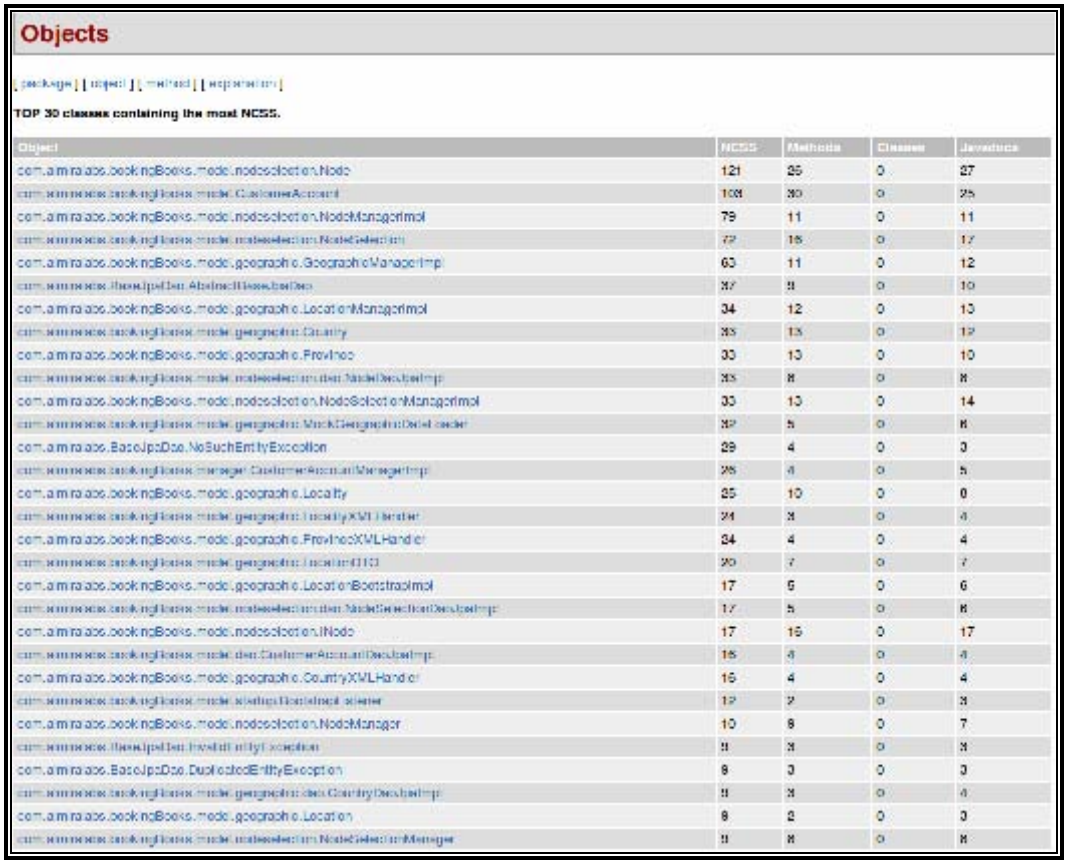
Objects

[package] [object] [method] [explanation]

TOP 30 classes containing the most NCSS.

Object	NCSS	Methods	Classes	Javadocs
com.amrraabe.bookingBooks.model.nodeselection.Node	121	35	0	27
com.amrraabe.bookingBooks.model.CustomerAccount	109	30	0	25
com.amrraabe.bookingBooks.model.nodeselection.NodeManagerImpl	79	11	0	11
com.amrraabe.bookingBooks.model.nodeselection.NodeSelection	72	16	0	17
com.amrraabe.bookingBooks.model.geographic.GeographicManagerImpl	63	11	0	12
com.amrraabe.BaseJpaDao.AbstractBaseDao	37	9	0	10
com.amrraabe.bookingBooks.model.geographic.LocationManagerImpl	34	12	0	13
com.amrraabe.bookingBooks.model.geographic.Country	33	13	0	12
com.amrraabe.bookingBooks.model.geographic.Province	33	13	0	10
com.amrraabe.bookingBooks.model.nodeselection.dao.NodeDaoImpl	33	8	0	8
com.amrraabe.bookingBooks.model.nodeselection.NodeSelectionManagerImpl	33	13	0	14
com.amrraabe.bookingBooks.model.geographic.WorldGeographicDataReader	32	5	0	6
com.amrraabe.BaseJpaDao.NoSuchEntityException	29	4	0	3
com.amrraabe.bookingBooks.manager.CustomerAccountManagerImpl	26	4	0	5
com.amrraabe.bookingBooks.model.geographic.Locality	25	10	0	9
com.amrraabe.bookingBooks.model.geographic.LocalityXMLHandler	24	3	0	4
com.amrraabe.bookingBooks.model.geographic.ProvinceXMLHandler	24	4	0	4
com.amrraabe.bookingBooks.model.geographic.LocationUtil	20	7	0	7
com.amrraabe.bookingBooks.model.geographic.LocationBootstrapImpl	17	5	0	6
com.amrraabe.bookingBooks.model.nodeselection.dao.NodeSelectionDaoImpl	17	5	0	6
com.amrraabe.bookingBooks.model.nodeselection.INode	17	16	0	17
com.amrraabe.bookingBooks.model.dao.CustomerAccountDaoImpl	16	4	0	4
com.amrraabe.bookingBooks.model.geographic.CountryXMLHandler	16	4	0	4
com.amrraabe.bookingBooks.model.xstream.XstreamImpl.clone	12	2	0	3
com.amrraabe.bookingBooks.model.nodeselection.NodeManager	10	8	0	7
com.amrraabe.BaseJpaDao.InvalidEntityException	9	3	0	3
com.amrraabe.BaseJpaDao.DuplicatedEntityException	9	3	0	3
com.amrraabe.bookingBooks.model.geographic.dao.CountryDaoImpl	9	3	0	4
com.amrraabe.bookingBooks.model.geographic.Location	9	2	0	3
com.amrraabe.bookingBooks.model.nodeselection.NodeSelectionManager	9	8	0	8

Picture III-71: JavaNCSS Metric Results Report (III)



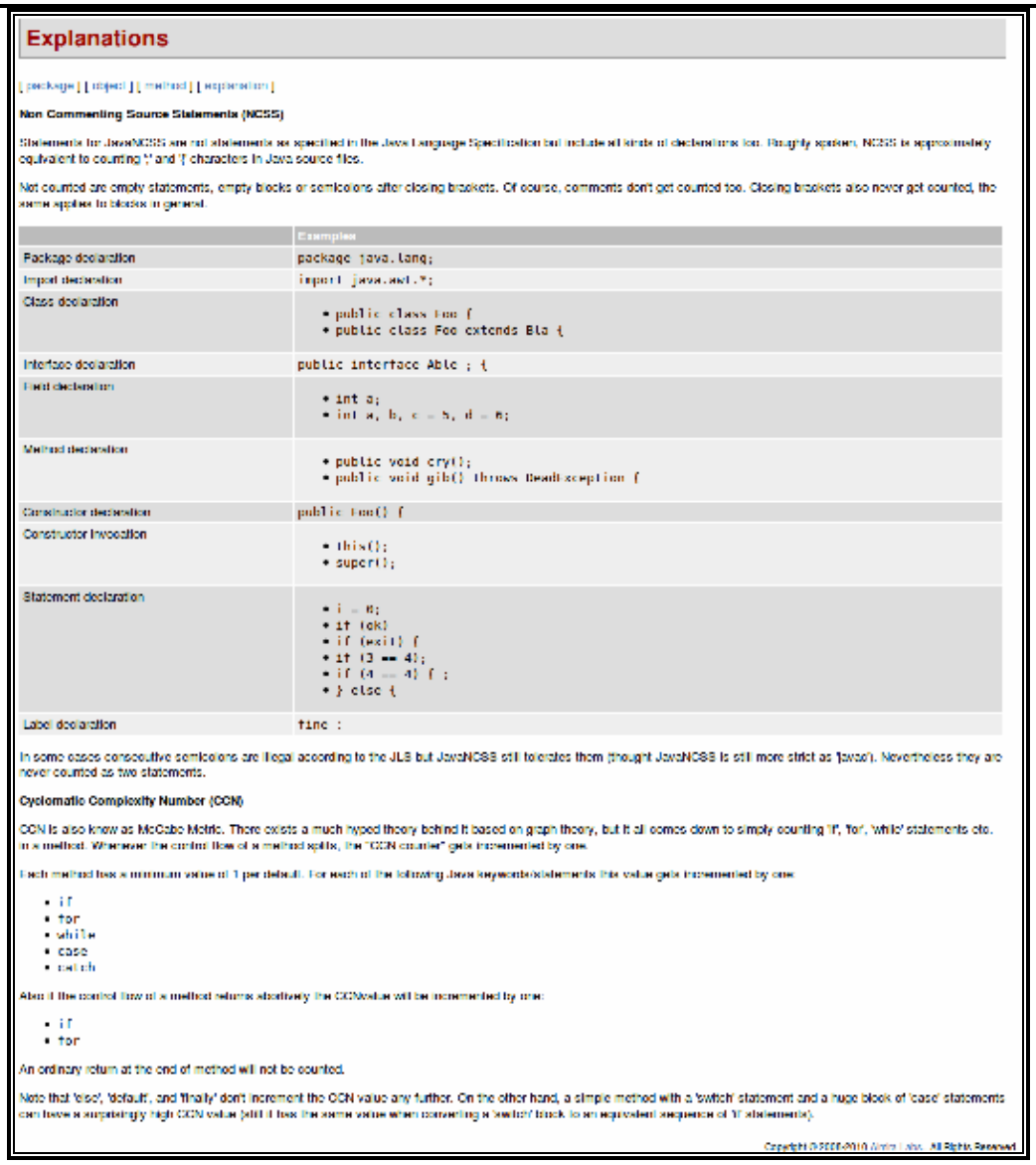
Objects

[package] [object] [method] [explanation]

TOP 30 classes containing the most NCSS.

Object	NCSS	Methods	Classes	Javadocs
com.amrraabe.bookingBooks.model.nodoseleccion.Nodo	121	35	0	27
com.amrraabe.bookingBooks.model.CuentaAccount	108	30	0	25
com.amrraabe.bookingBooks.model.nodoseleccion.NodoManagerImpl	79	11	0	11
com.amrraabe.bookingBooks.model.nodoseleccion.NodoSeleccion	72	16	0	17
com.amrraabe.bookingBooks.model.geografico.GeograficoManagerImpl	63	11	0	12
com.amrraabe.BaseJpaDao.AbstractBaseJpaDao	37	8	0	10
com.amrraabe.bookingBooks.model.geografico.LocationManagerImpl	34	12	0	13
com.amrraabe.bookingBooks.model.geografico.Country	33	13	0	12
com.amrraabe.bookingBooks.model.geografico.Province	30	13	0	10
com.amrraabe.bookingBooks.model.nodoseleccion.dao.NodoDaoImpl	28	8	0	8
com.amrraabe.bookingBooks.model.nodoseleccion.NodoSeleccionManagerImpl	30	13	0	14
com.amrraabe.bookingBooks.model.geografico.MockGeograficoDataLoader	29	5	0	6
com.amrraabe.BaseJpaDao.NoSuchEntityException	29	4	0	3
com.amrraabe.bookingBooks.manager.CuentaAccountManagerImpl	26	4	0	5
com.amrraabe.bookingBooks.model.geografico.Locality	25	10	0	8
com.amrraabe.bookingBooks.model.geografico.LocalityXMLHandler	24	3	0	4
com.amrraabe.bookingBooks.model.geografico.ProvinceXMLHandler	24	4	0	4
com.amrraabe.bookingBooks.model.geografico.Location4113	20	7	0	7
com.amrraabe.bookingBooks.model.geografico.LocationBootstrapImpl	17	6	0	6
com.amrraabe.bookingBooks.model.nodoseleccion.dao.NodoSeleccionDaoImpl	17	5	0	6
com.amrraabe.bookingBooks.model.nodoseleccion.INodo	17	16	0	17
com.amrraabe.bookingBooks.model.dao.CuentaAccountDaoImpl	16	4	0	4
com.amrraabe.bookingBooks.model.geografico.CountryXMLHandler	16	4	0	4
com.amrraabe.bookingBooks.model.xmlbootstrap.BootstrapLoader	12	2	0	3
com.amrraabe.bookingBooks.model.nodoseleccion.NodoManager	10	6	0	7
com.amrraabe.BaseJpaDao.InvalidEntityException	9	3	0	3
com.amrraabe.BaseJpaDao.DuplicatedEntityException	9	3	0	3
com.amrraabe.bookingBooks.model.geografico.dao.CountryDaoImpl	9	3	0	4
com.amrraabe.bookingBooks.model.geografico.Location	8	2	0	3
com.amrraabe.bookingBooks.model.nodoseleccion.NodoSeleccionManager	8	6	0	6

Picture III-72: JavaNCSS Metric Results Report (IV)



Explanations

[package] [object] [method] [expression]

Non Commenting Source Statements (NCSS)

Statements for JavaNCSS are not statements as specified in the Java Language Specification but include all kinds of declarations too. Roughly spoken, NCSS is approximately equivalent to counting ';' and '{' characters in Java source files.

Not counted are empty statements, empty blocks or semicolons after closing brackets. Of course, comments don't get counted too. Closing brackets also never get counted, the same applies to blocks in general.

	Examples
Package declaration	`package java.lang;`
Import declaration	`import java.awt.*;`
Class declaration	`public class Foo {` `public class Foo extends Bla {`
Interface declaration	`public interface Able ; {`
Field declaration	`int a;` `int a, b, c = 5, d = 6;`
Method declaration	`public void cry();` `public void gib() throws DeadException {`
Constructor declaration	`public Foo() {`
Constructor invocation	`this();` `super();`

Statement declaration	`i = 0;`
	`if (ok)`
	`if (exit) {`
	`if (3 == 4);`
	`if (4 == 4) { ;`
	`} else {`

Label declaration	`fine :`

In some cases consecutive semicolons are illegal according to the JLS but JavaNCSS still tolerates them (though JavaNCSS is still more strict as 'javac'). Nevertheless they are never counted as two statements.

Cyclomatic Complexity Number (CCN)

CCN is also know as McCabe Metric. There exists a much hyped theory behind it based on graph theory, but it all comes down to simply counting 'if', 'for', 'while' statements etc. in a method. Whenever the control flow of a method splits, the "CCN counter" gets incremented by one.

Each method has a minimum value of 1 per default. For each of the following Java keywords/statements the value gets incremented by one:

- `if`
- `for`
- `while`
- `case`
- `catch`

Also if the control flow of a method returns abruptly the CCN value will be incremented by one:

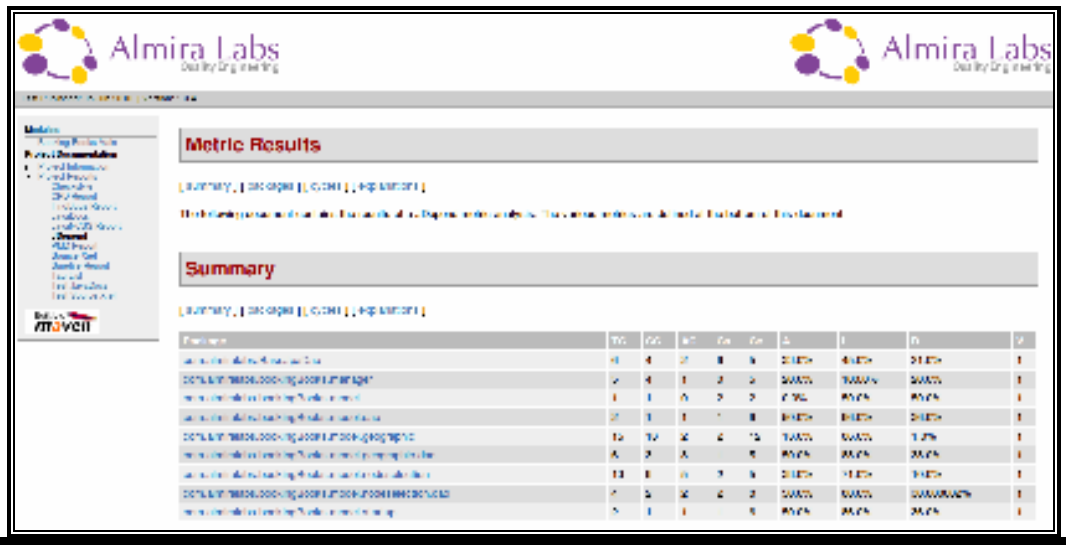
- `if`
- `for`

An ordinary return at the end of method will not be counted.

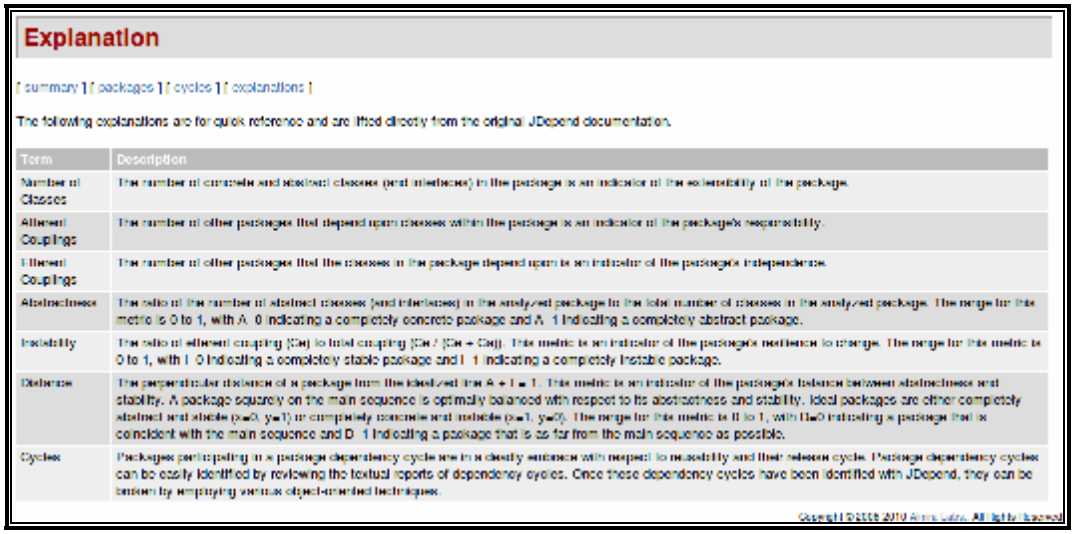
Note that 'else', 'default', and 'finally' don't increment the CCN value any further. On the other hand, a simple method with a 'switch' statement and a huge block of 'case' statements can have a surprisingly high CCN value (still it has the same value when converting a 'switch' block to an equivalent sequence of 'if' statements).

Copyright © 2006-2010 Jens Lehr. All Rights Reserved.

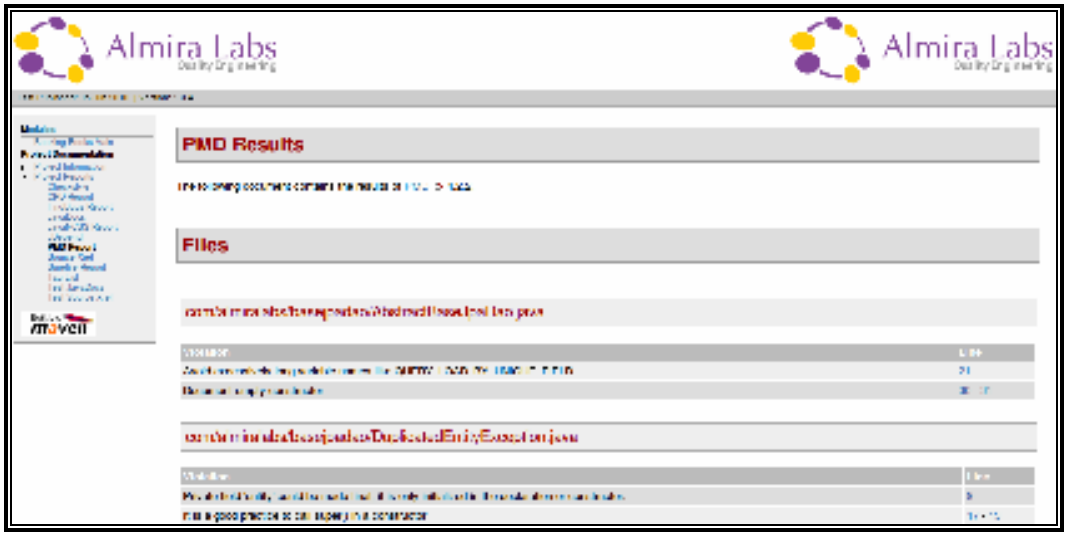
Picture III-73: JavaNCSS Metric Results Report (V)



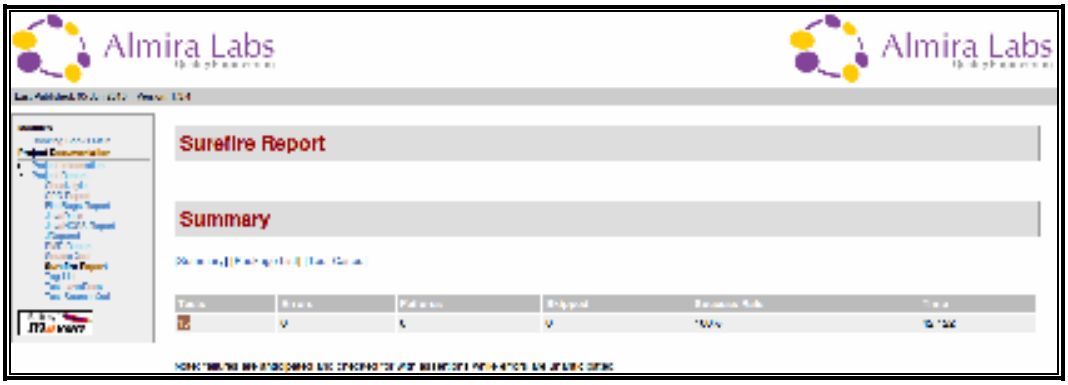
Picture III-74: JDependt Report (I)



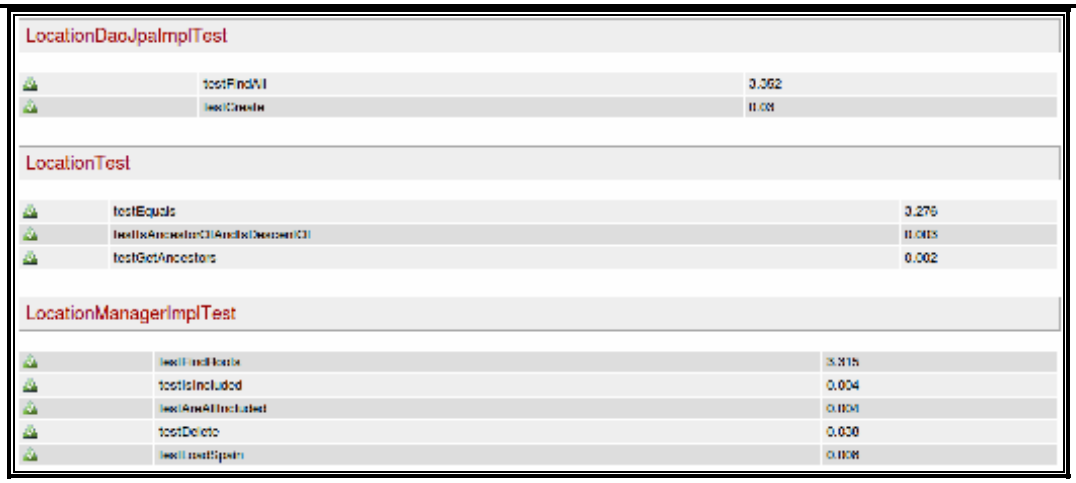
Picture III-75: JDependt Report (II)



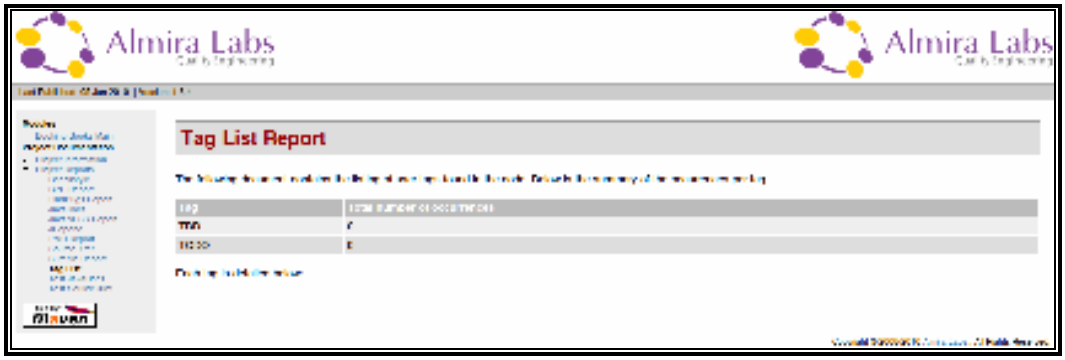
Picture III-76: PMD Results



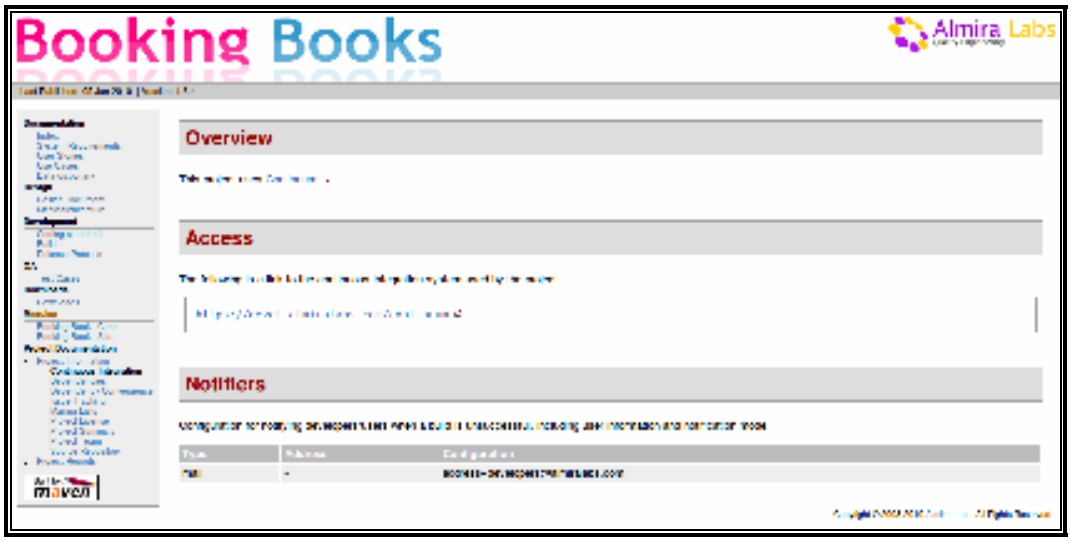
Picture III-77: Surefire Report (I)



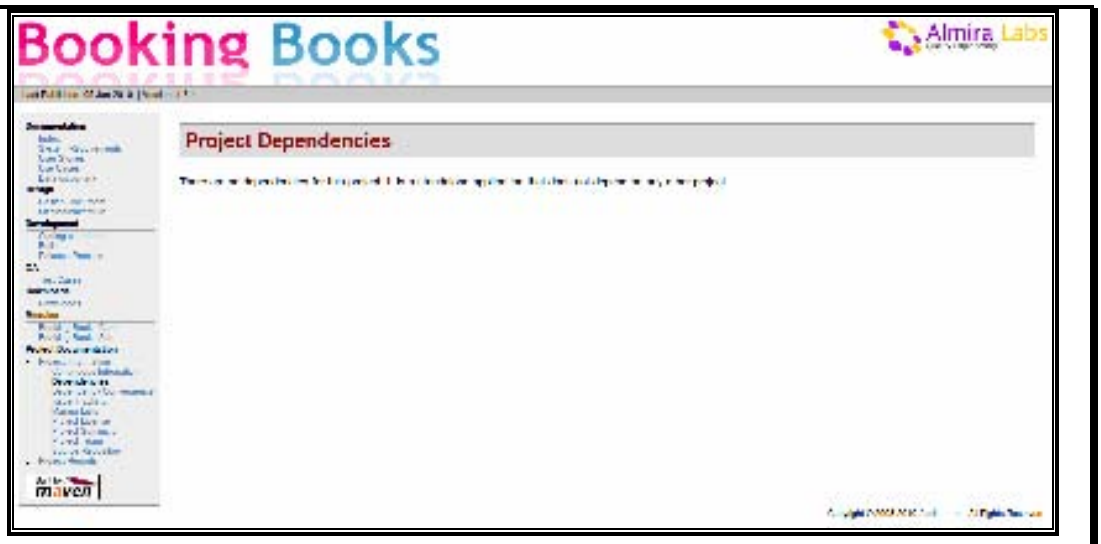
Picture III-78: Surefire Report (II)



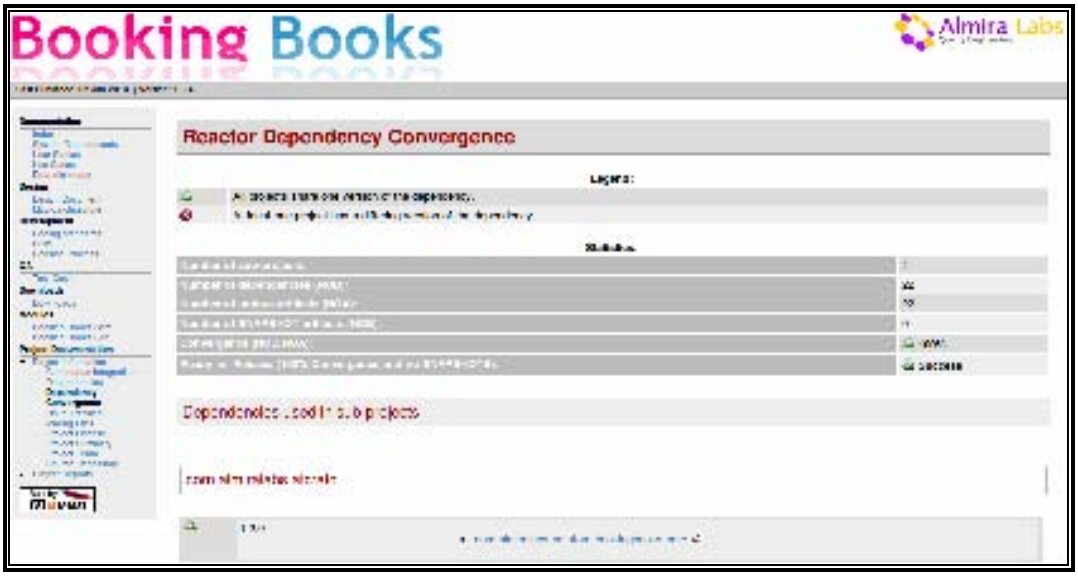
Picture III-79: Tag List Report



Picture III-80: Main Project Information Report

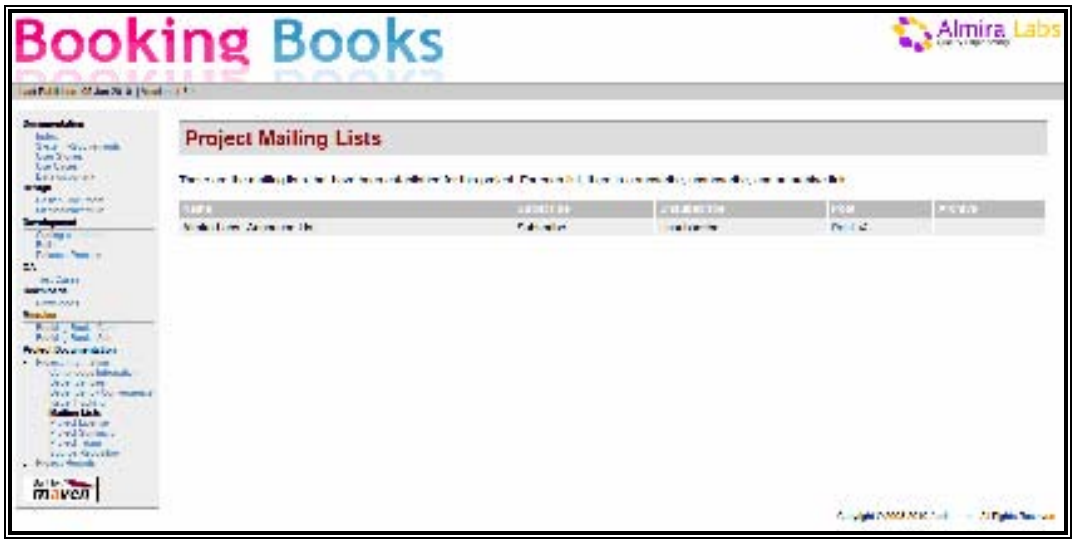


Picture III-81: Project Dependencies Report

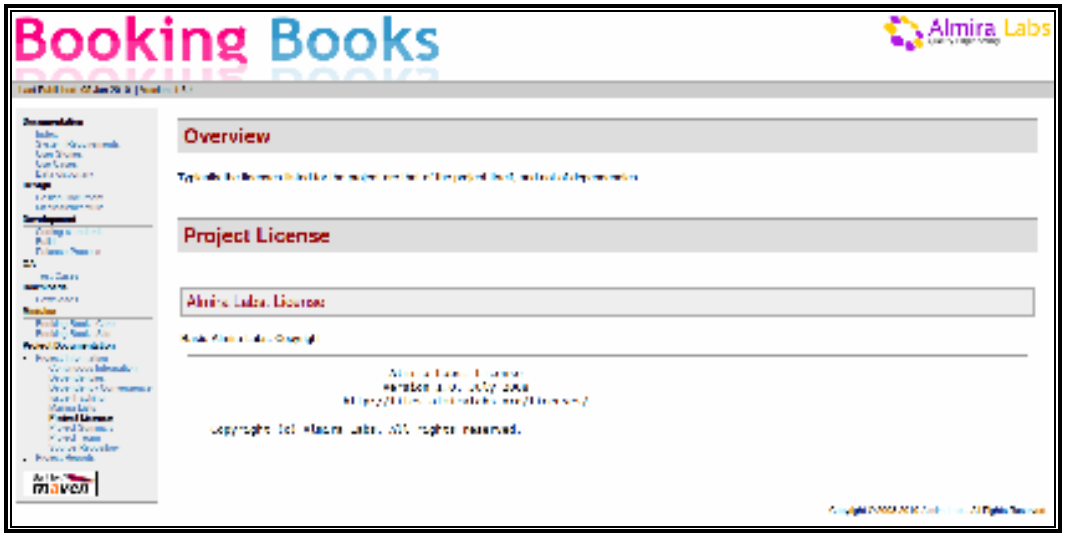


Picture III-82: Reactor Dependency Convergence Report

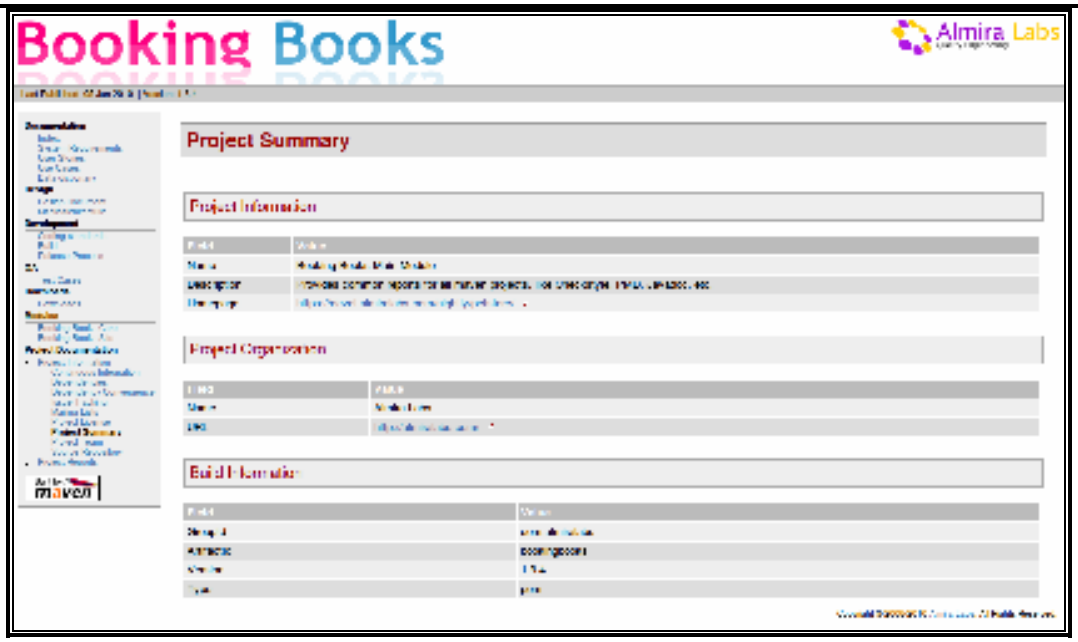
Picture III-83: Issue Tracking Report



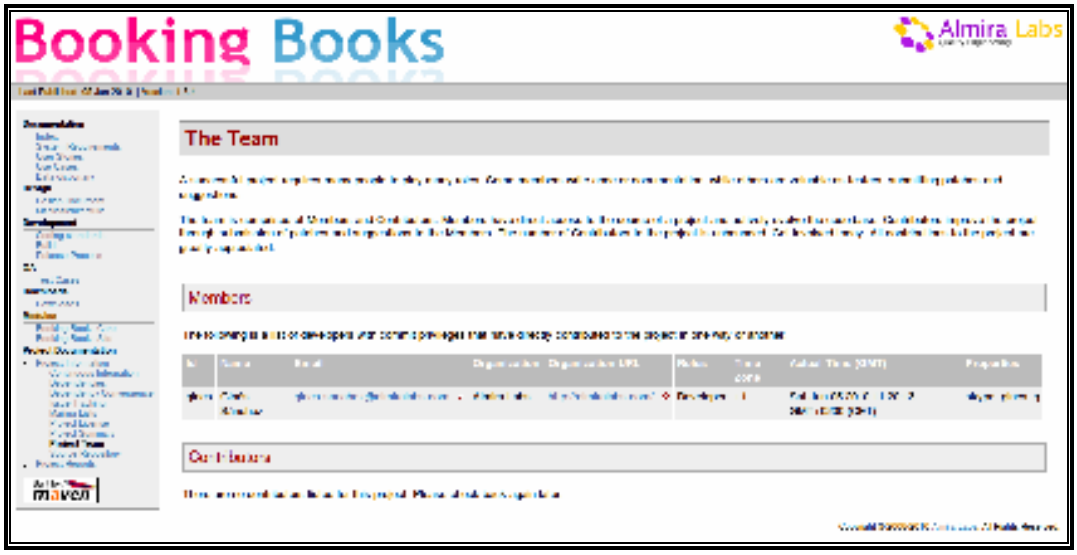
Picture III-84: Mailing List Report



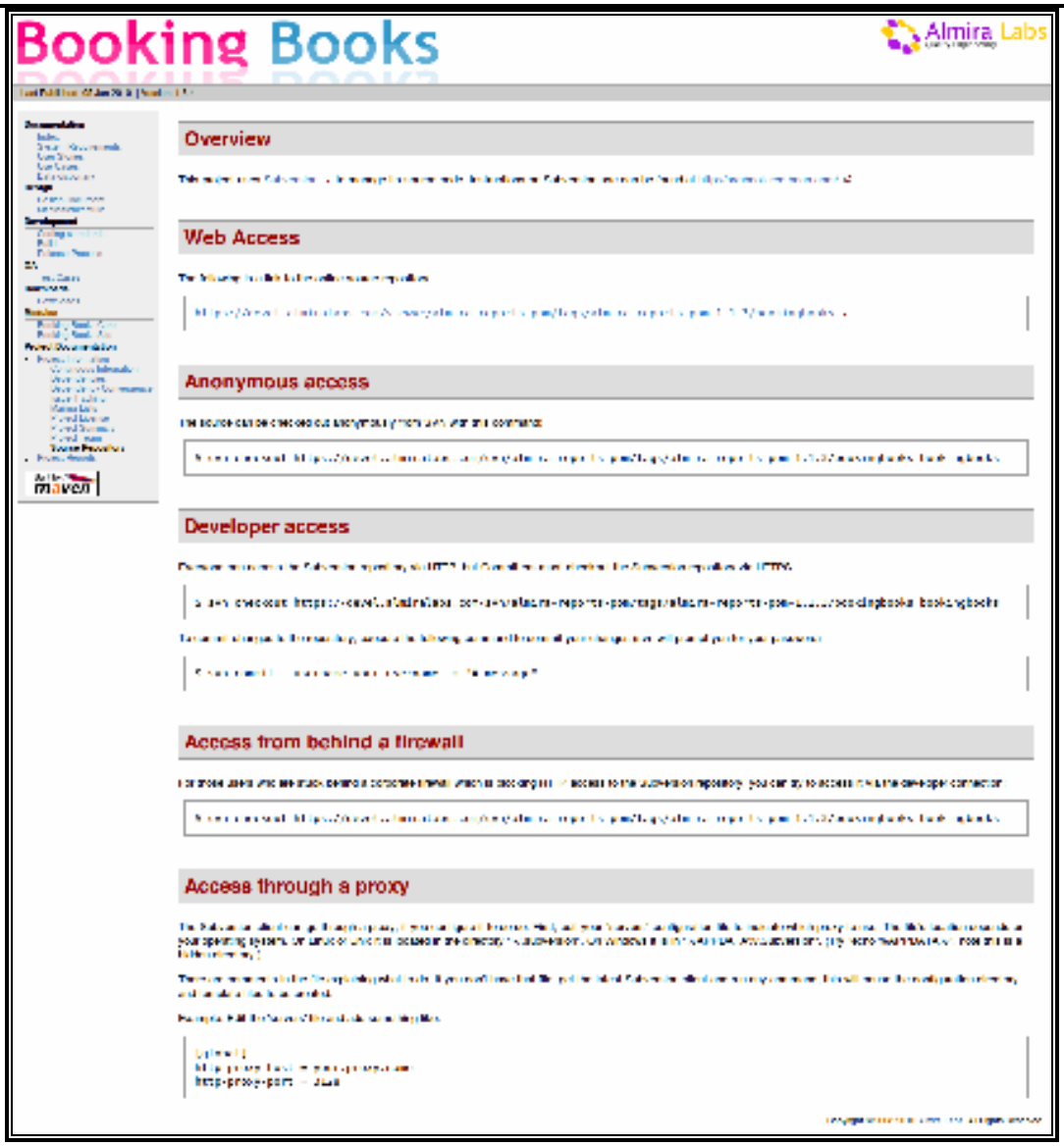
Picture III-85: Project License Report



Picture III-86: Project Summary Report



Picture III-87: Project Team Report



Picture III-88: Source Repository Report

7.3.4. Design and development review

ISO 9001:2008 Standard

At suitable stages, systematic reviews of design and development shall be performed in accordance with planned arrangements (see 7.3.1)

a) to evaluate the ability of the results of design and development to meet requirements, and

b) to identify any problems and propose necessary actions.

Participants in such reviews shall include representatives of functions concerned with the design and development stage(s) being

reviewed. Records of the results of the reviews and any necessary actions shall be maintained (see 4.2.4).

ISO 90003:2004 Guideline

The degree of formality and rigor of the activities associated with the review processes should be appropriate for the complexity of the product, the quality requirements and the degree of risk associated with the specified use of the software product. The organization should establish procedures for dealing with process and product deficiencies or nonconformities identified during these activities (see 8.3). It is recommended that these procedures be documented.

During design and development reviews, criteria such as feasibility, security, safety, programming rules and testability should be taken into account.

Review of design and development should be performed in accordance with planned arrangements. The elements of the review to be considered are the following:

a) what is to be reviewed, when and the type of review, such as demonstrations, formal proof of correctness, inspections, walkthroughs and joint reviews;

b) what functional groups would be concerned in each type of review and, if there is to be a review meeting, how it is to be organized and conducted;

c) what records have to be produced, e.g. meeting minutes, issues, problems, actions and action status;

d) the methods for monitoring the application of rules, practices and conventions to ensure requirements are met;

e) what has to be done prior to the conduct of a review, such as establishment of objectives, meeting agenda, documents required and roles of review personnel;

f) what has to be done during the review, including the techniques to be used and guidelines for all participants;

g) the success criteria for the review;

h) what follow-up activities are used to ensure that issues identified at the review are resolved.

Further design and development activities should proceed only when the consequences of all known deficiencies are understood, or the risk of proceeding otherwise is known and agreed. Any findings should be addressed and resolved, as appropriate.

NOTE 2 For further information, see the following:

— ISO/IEC 12207:1995[11], 5.3.4.2, 5.3.5.6 and 5.3.6.7 (requirements and design evaluations) and 6.6.3 (technical reviews), and ISO/IEC 12207:1995/Amd.1:2002[12], F.2.6 (joint reviews);

— ISO/IEC TR 15271:1998[21], Annex A (quality processes and evaluation requirements).

Guideline Application

1. Description

The reviews depend of the issue type (for example, a bug, a use case specification, a use case implementation, etc). Not all the issues must be reviewed. One review for every issue type every month is enough.

Guideline Application Tool

1. Description

The reviews must be done aver issue types in *Jira*.

7.3.5. Design and development verification

ISO 9001:2008 Standard

Verification shall be performed in accordance with planned arrangements (see 7.3.1) to ensure that the design and development outputs have met the design and development input requirements. Records of the results of the verification and any necessary actions shall be maintained (see 4.2.4).

ISO 90003:2004 Guideline

Verification of software is aimed at providing assurance that the output of a design and development activity conforms to the input requirements.

Verification should be performed as appropriate during design and development. Verification may comprise reviews of design and development output (e.g. by inspections and walkthroughs), analysis, demonstrations including prototypes, simulations or tests. Verification

may be conducted on the output from other activities, e.g. COTS, purchased and customer-supplied products.

The verification results and any further actions should be recorded and checked when the actions are completed.

When the size, complexity or criticality of a software product warrants, specific assurance methods should be used for verification, such as complexity metrics, peer reviews, condition/decision coverage or formal methods.

Only verified design and development outputs should be submitted for acceptance and subsequent use. Any findings should be addressed and resolved, as appropriate.

NOTE For further information, see ISO/IEC 12207:1995[11], 5.3 (development) and 6.4 (verification), and ISO/IEC 12207:1995/Amd.1:2002[12], F.1.3 (development) and F.2.4 (verification).

Guideline Application

1. Description

The issue design and development verification is included in the design and development reviews, so it is not necessary add any guideline.

Guideline Application Tool

1. Description

It is not necessary add any guideline application tool.

7.3.6. Design and development validation

ISO 90003:2004 Guideline

Design and development validation shall be performed in accordance with planned arrangements (see 7.3.1) to ensure that the resulting product is capable of meeting the requirements for the specified application or intended use, where known. Wherever practicable, validation shall be completed prior to the delivery or implementation of the product. Records of the results of validation and any necessary actions shall be maintained (see 4.2.4).

7.3.6.1. Validation

ISO 90003:2004 Guideline

Validation of software is aimed at providing reasonable confidence that it will meet its operational requirements.

Before offering the product for customer acceptance, the organization should validate the operation of the product in accordance with its specified intended use, under conditions similar to the application environment, as specified in the contract. Any differences between the validation environment and the actual application environment, and the risks associated with such differences, should be identified and justified as early in the life cycle as possible, and recorded. In the course of validation, configuration audits or evaluations may be performed, where appropriate, before release of a configuration baseline. Configuration audits or evaluations confirm, by examination of the review, inspection and test records, that the software product complies with its contractual or specified requirements. This may require analysis, simulation or emulation where validation is not practicable in operational conditions.

In software development, it is important that the validation results and any further actions required to meet the specified requirements are recorded, and checked when the actions are completed.

In some cases, it may not be possible, or feasible, to validate fully the software product by measurement and monitoring. An example may be where safety-related software cannot be tested under actual circumstances without risking serious consequences, or perhaps the actual circumstances themselves are rare and difficult to simulate.

The inability to test some software products exhaustively and conclusively may lead the organization to decide

a) how confidence can be gained from the development and tools used, and

b) what types of testing or analysis can be performed to increase confidence that the product will perform correctly under the *untesetable* circumstances, e.g. static code analysis.

Whatever methods are used, they should be commensurate with the risk and consequences of design and development failures.

Guideline Application

1. Description

Validation means confirmation; therefore, in the context of Software, the validation process means confirmation that the

requirements baseline functions and performances are correctly and completely implemented in the final product.

The validation must ensure that the following aspects will be addressed during the validation of the software development in the context of the project:

- Forward traceability; is included in Xuse in the Document Information page.
- Backwards traceability; is included in Xuse, in the Document Information page, too.
- Functional audits; not applicable for this project.
- Documentation; is included in Xuse, code comments and so on.
- Testing; is included in the test part.
- Non-functional requirements; are included in Xuse.
- Code evaluation; is included in Xuse as a meeting minute.
- Integration test plan; included as integration test.
- Unit test plan; included as unitary test.
- Test cases description; included in Xuse as a template.

Guideline Application Tool

1. Description

It is not necessary add any guideline application tool.

7.3.6.2. Testing

ISO 90003:2004 Guideline

Validation may often be performed by testing. Testing may be required at several levels, from the individual software item to the complete software product. There are several different approaches to testing and the extent of testing and the degree of controls on the test environment, test inputs and test outputs may vary with the approach, the complexity of the product and the risk associated with the use of the product. Test planning should address test types, objectives, sequence and scope of testing, test cases, test data and expected results. Test planning should identify the human and physical resources needed for testing and define the responsibilities of those involved.

Specific testing for software includes establishing, documenting, reviewing and implementing plans for the following:

a) unit tests, i.e. stand-alone tests of software components;

b) integration and system tests, i.e. tests of aggregations of software components (and the complete system);

c) qualification tests, i.e. tests of the complete software product prior to delivery to confirm the software meets its defined requirements;

d) acceptance tests, i.e. tests of the complete software product to confirm the software meets its acceptance criteria.

Regression testing should be performed to verify or validate that the capabilities of the software have not been compromised by a change.

Acceptance tests are those that are performed for the customer's benefit with the aim of determining the acceptability of the product. Acceptance may be with or without defects or deviations from requirements, by agreement of the parties involved.

Testing tools and the environment to be used should be qualified and controlled, and any limitations to testing recorded.

Testing procedures should cover recording and analysis of results as well as problem and change management.

NOTE For further information, see the following:

— ISO/IEC 12207:1995[11], 5.3 (development) and 6.5 (validation), and ISO/IEC 12207:1995/Amd.1:2002[12], F.1.3 (development) and F.2.5 (validation);

— ISO/IEC 14598-3[15] and ISO/IEC 14598-5[17]

Guideline Application

1. Description

I have done unitary test (over DAOs, Managers, *Struts Actions*), integration test and unit acceptance test (UAT) in this Project. NOTE: Unitary Test source code is in chapter *IV - Booking Books Application* section *4. Source Code.*

2. Unitary Test

A unitary test is a software verification and validation method in which a developer test if individual units of source code are fit for use. A unit is the smallest testable part of an application.

Ideally, each test is independent from the others; for example, when testing a Manager, we shouldn’t test the part of which depends (a DAO, for example). Unit tests are typically written and

run by software developers to ensure that code meets its design and behaves as intended.

2.1. Testing the DAOs

When programming a use case must be created the Entity, and them the DAO. DAO are the part of the architecture in which are done the CRUD operations over the Entities, and if necessary the finder methods.

2.2. Testing the Managers

In the Manager is created the business logic. Managers use the DAO to perform their operations.

- **Test *CustomerAccountManagerImplTest***

In this example, I have created two different types of method to test the *CustomerAccountManager*; the class *CustomerAccountManagerImplTest* and the class *CustomerAccountManagerImplTestMock*.

Why I have done two tests for the same class? As I have said, the test must be independent from other architecture layers. In the class *CustomerAccountManagerImplTest*, I am testing the lower layers (the DAOs); It is a bad practice, because I am creating coupling (remember that in Object Oriented we always want high cohesion and low coupling) with other software artifacts (DAOs in this case). If one of the DAOs fails, the *CustomerAccountManagerImplTest* will fail, too.

That's the reason to create another class to test this Manager: *CustomerAccountManagerImplTestMock*. As you will see, in this class I am not using *JUnit* to test: I am using *EasyMock*. *EasyMock* provides Mock Objects for interfaces; it means that you set all Daos behaviour with mocks, and when Manager performs calls to Daos, they will have the behaviour previously defined.

2.3. Testing the Struts Actions

Struts is the de-facto Controller for J2EE which implements the MVC design pattern.

3. Integration Tests

Integration test is the phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before system testing. Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an

integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.

4. Unit Acceptance Tests

UAT is an acronym for **Unit Acceptance Test**, which confirms that the unit has been installed and that it is possible to communicate with the unit.

This test is a test of the entire architecture including routers, data backup devices, firewalls, etc. This will require the development of a testing protocol that may include testing scripts. The testing should be planned bearing in mind risk issues and contractual obligations.

Guideline Application Tool

1. Description

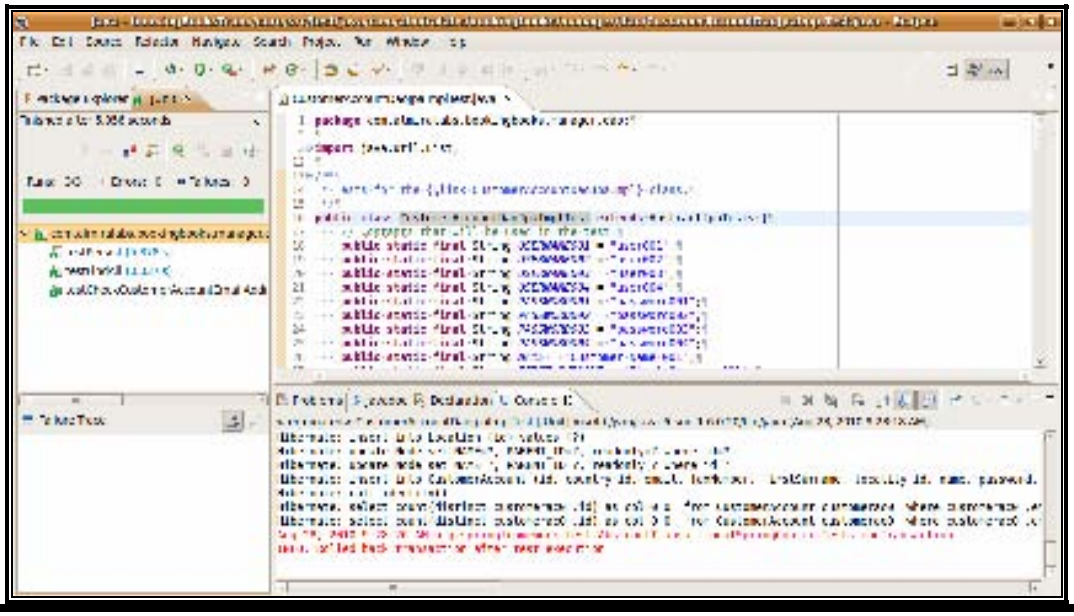
I will perform unitary test using *JUnit* and *EasyMock*; and integration test using Jmeter.

2. Unitary Test

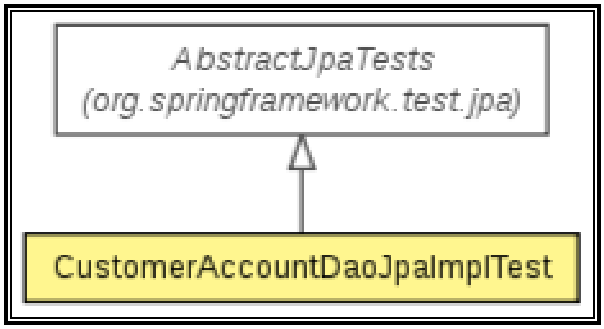
2.1. Testing the DAOs

- **Test *CustomerAccountDaoJpaImplTest***

CutomerAccountDao behaviour is tested in the class CustomerAccountDaoJpaImplTest, which extends of AbstractJpaTest.



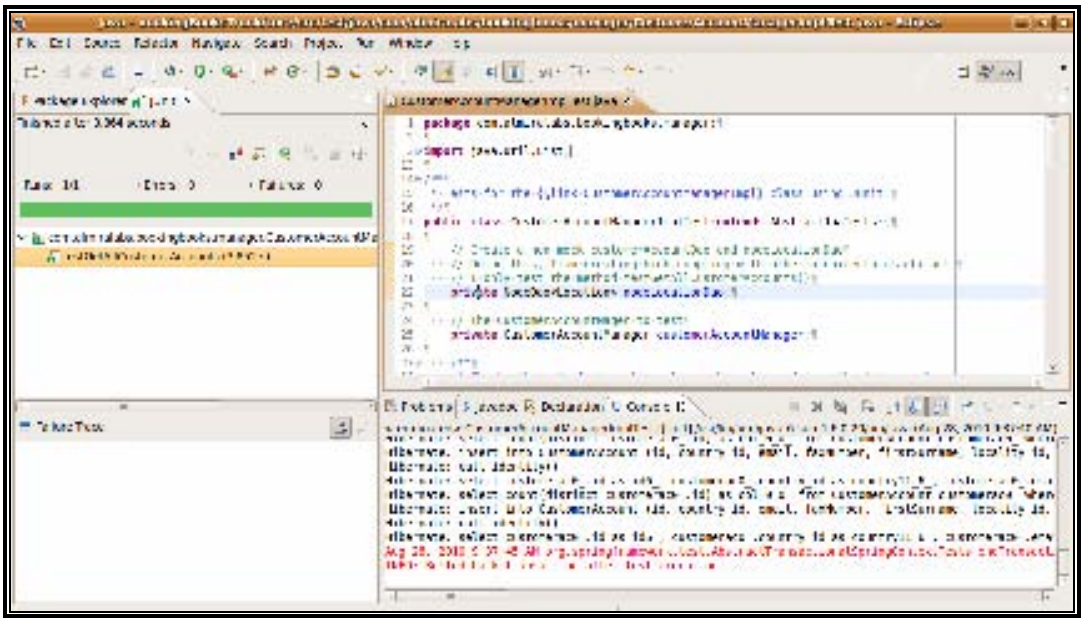
Picture III-89: CusomerAccountDaoJplaImplTest Eclipse Execution



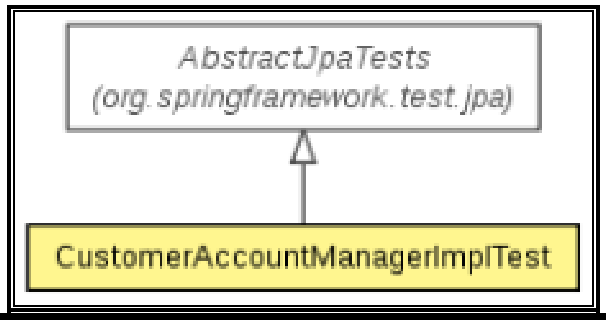
Picture III-90: CusomerAccountDaoJplaImplTest Class Diagram

2.2. Testing the Managers

- **Test *CustomerAccountManagerImplTest***

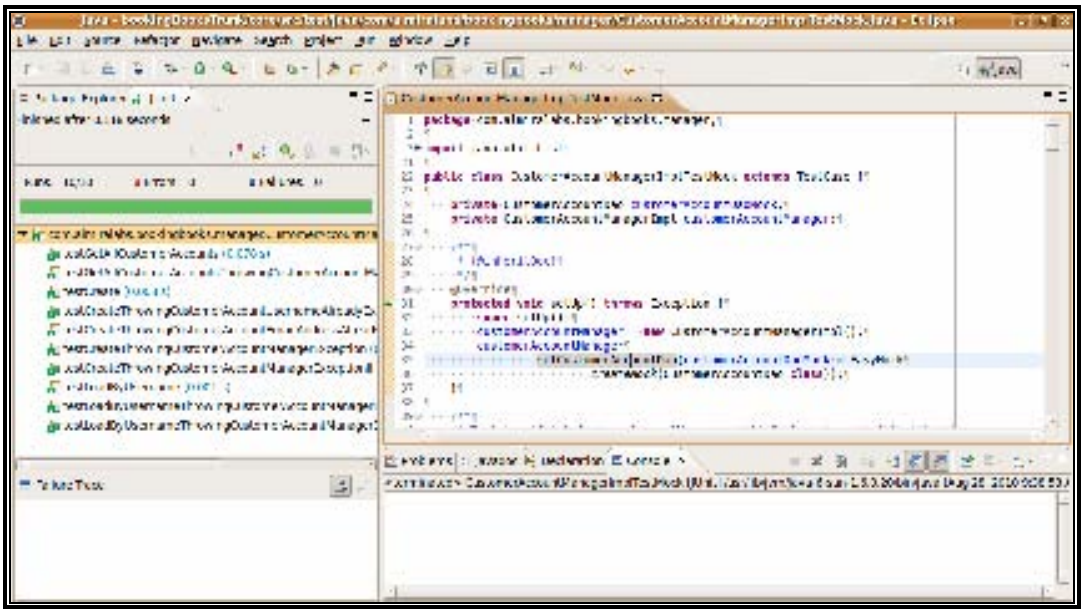


Picture III-91: CustomerAccountManagerImplTest Eclipse Execution

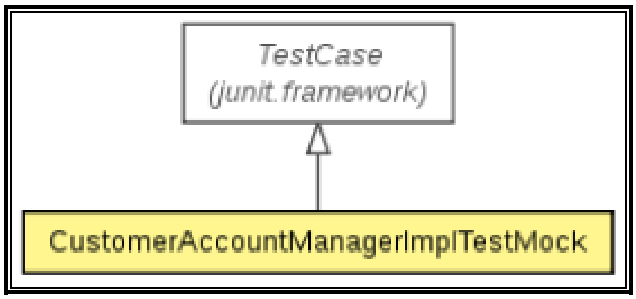


Picture III-92: CustomerAccountManagerImplTest Class Diagram

- **Test *CustomerAccountManagerImplTestMock***



Picture III-93: CustomerAccountImplTestMock Eclipse Execution

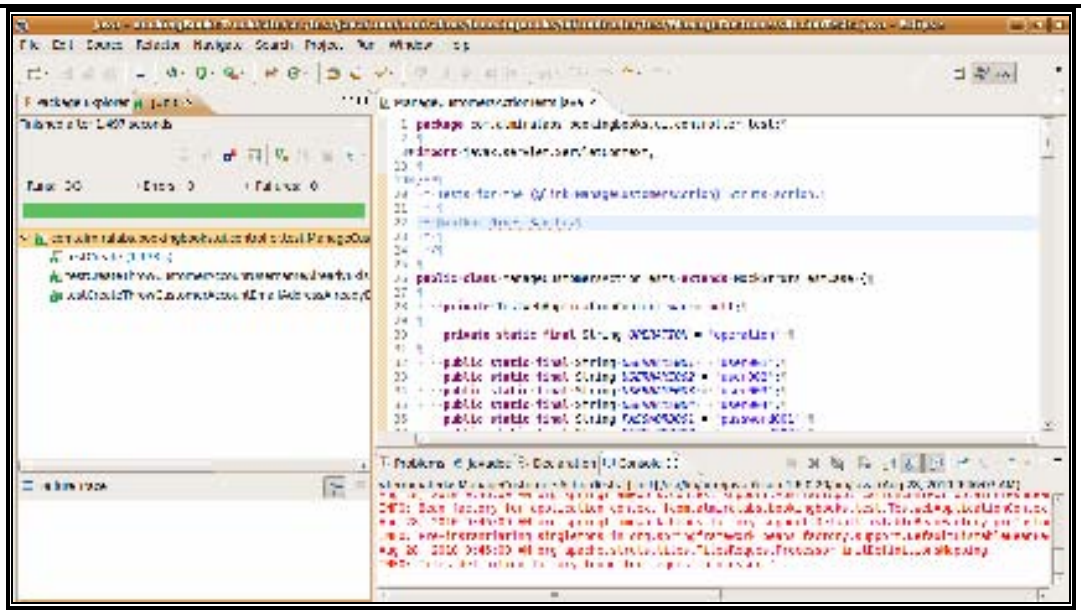


Picture III-94: CustomerAccountImplTestMock Class Diagram

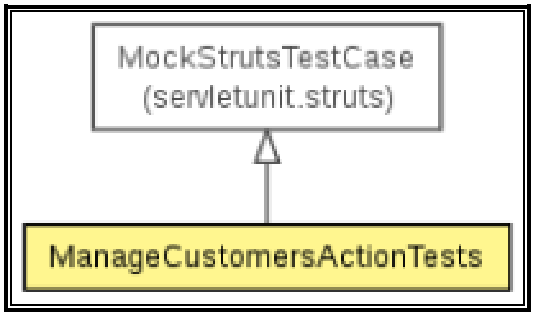
2.3. Testing the Struts Actions

To test the *Struts Actions*, I will use *EasyMock*.

- **Test *ManageCustomersActionTests***

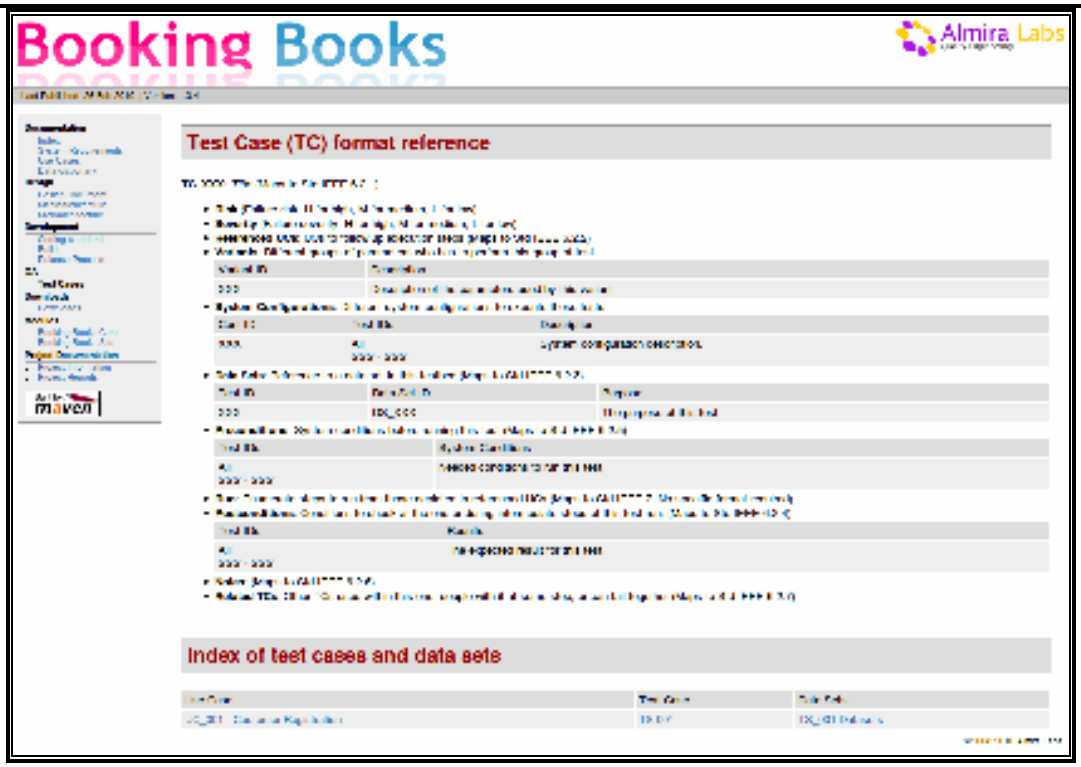


Picture III-95: ManageCustomersActionTests Eclipse Execution

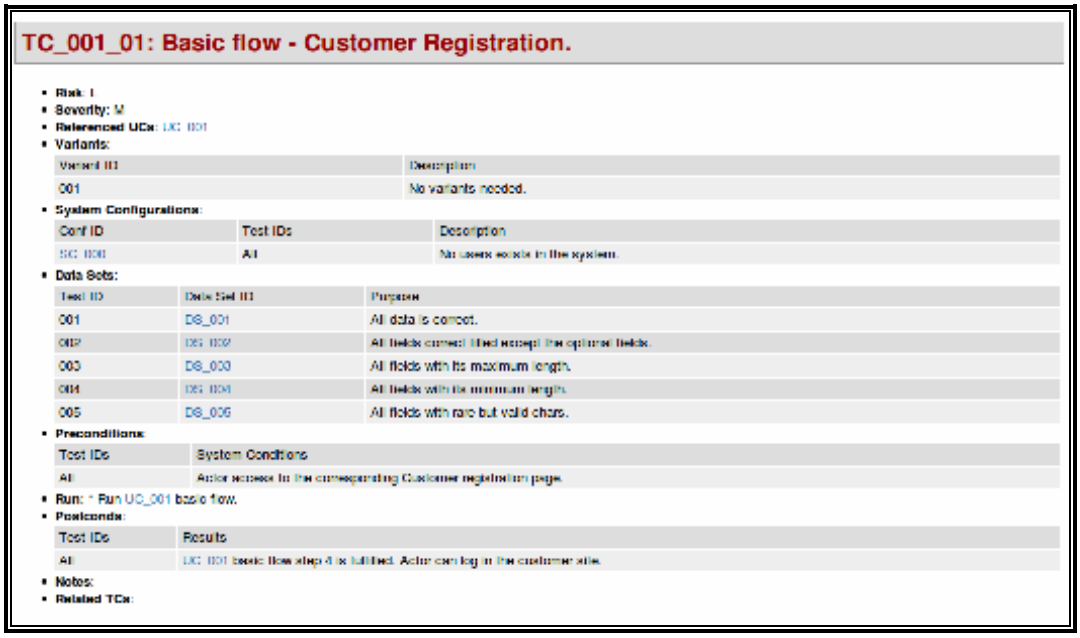


Picture III-96: ManageCustomersActionTests Class Diagram

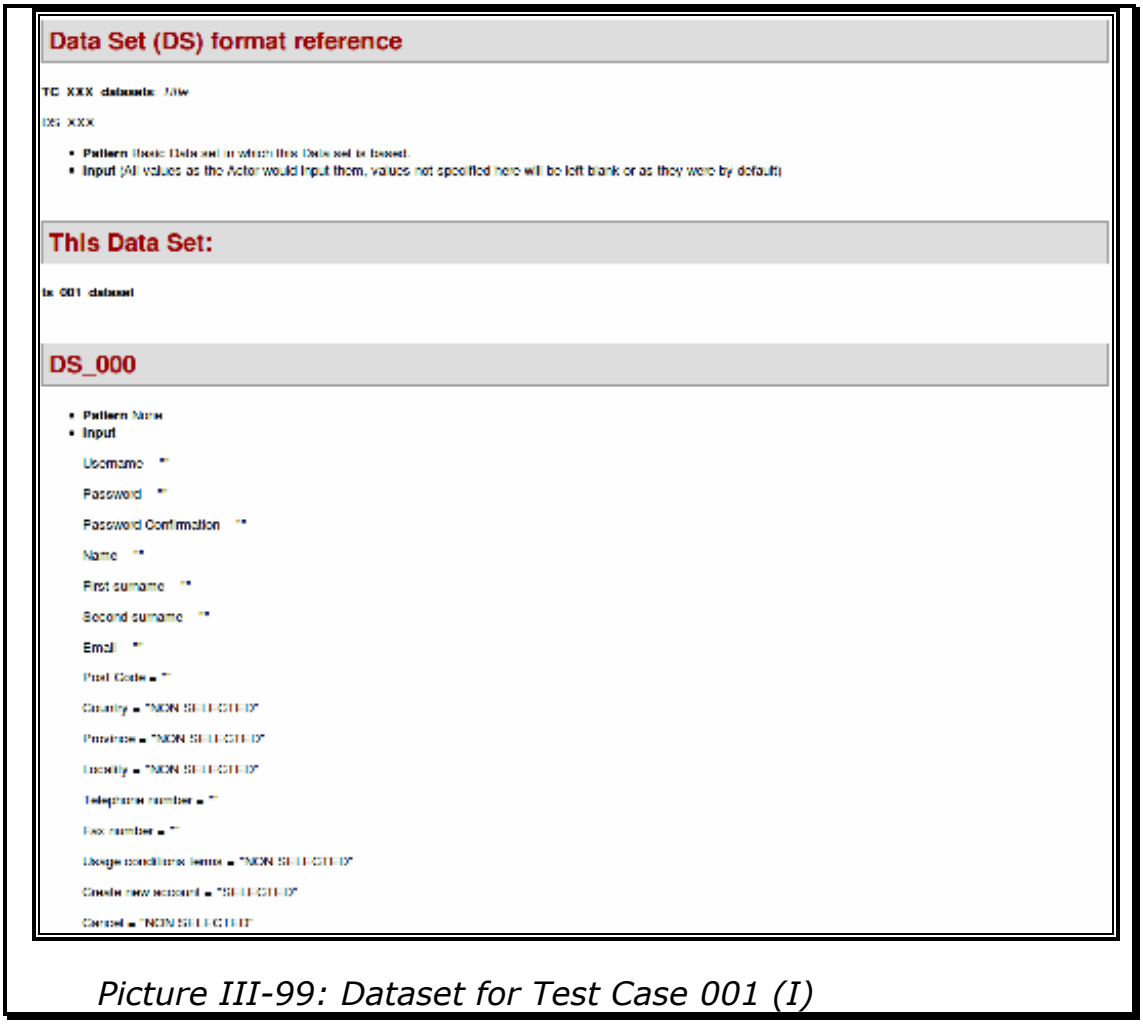
3. Integration Tests



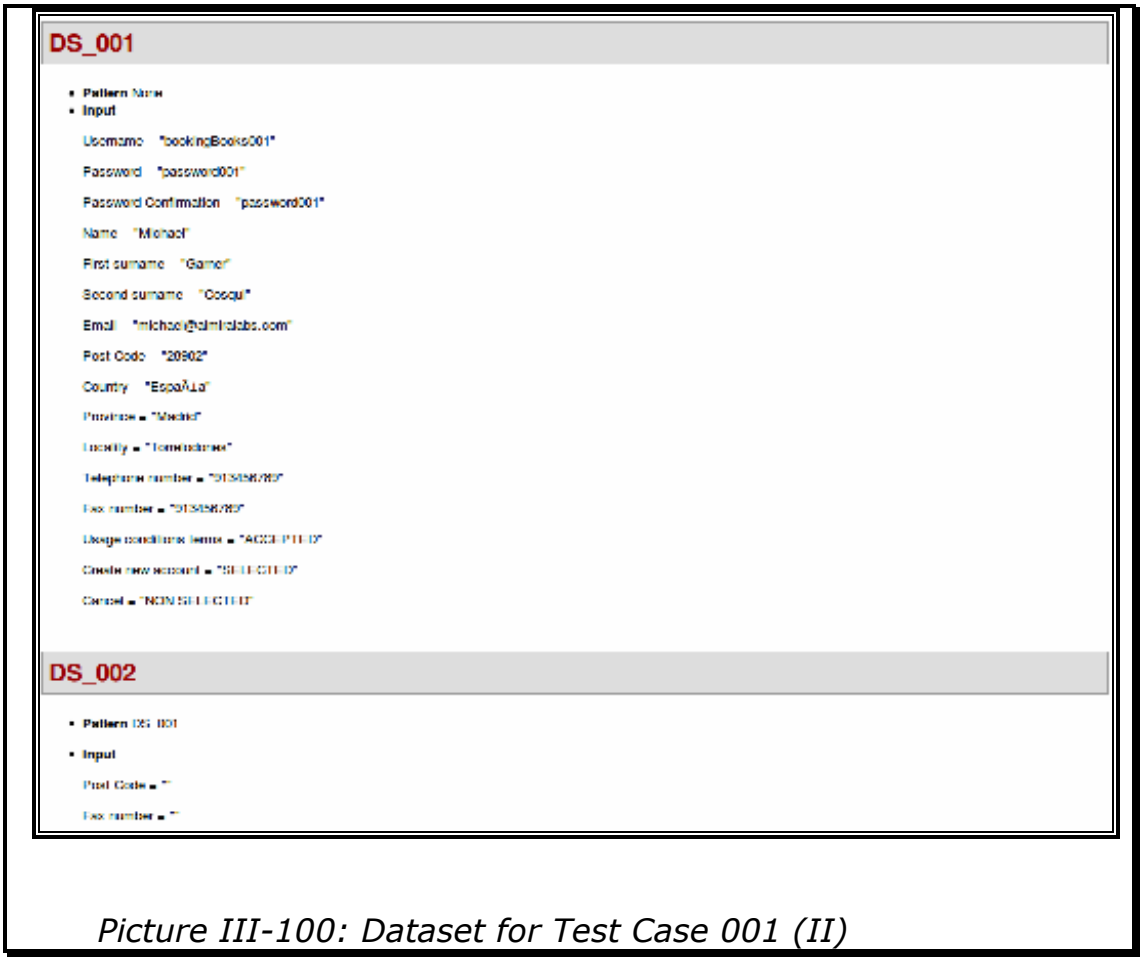
Picture III-97: Test Case Format Reference Diagram



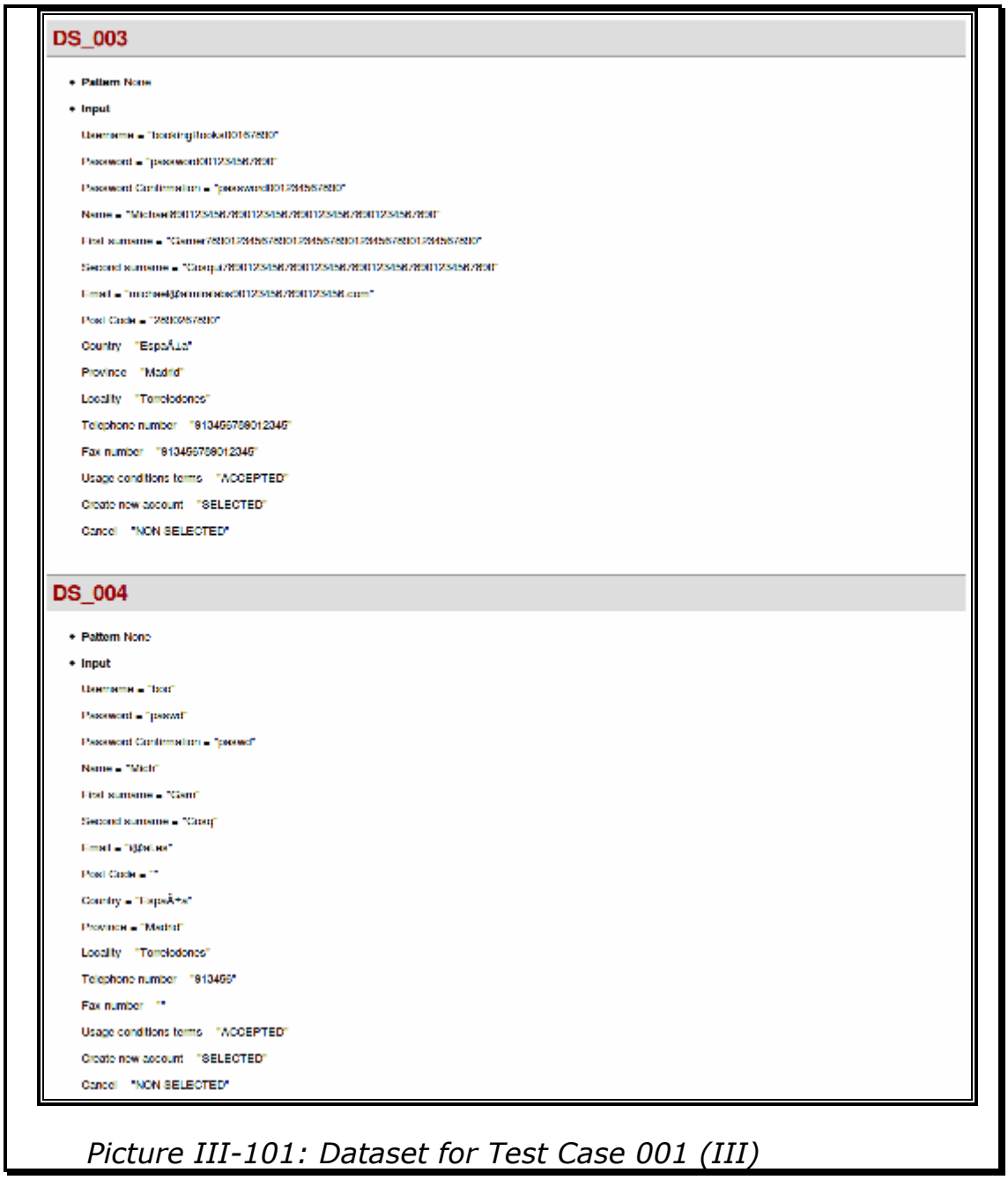
Picture III-98: Test Case 001_01: Basic Flow – Customer Registration



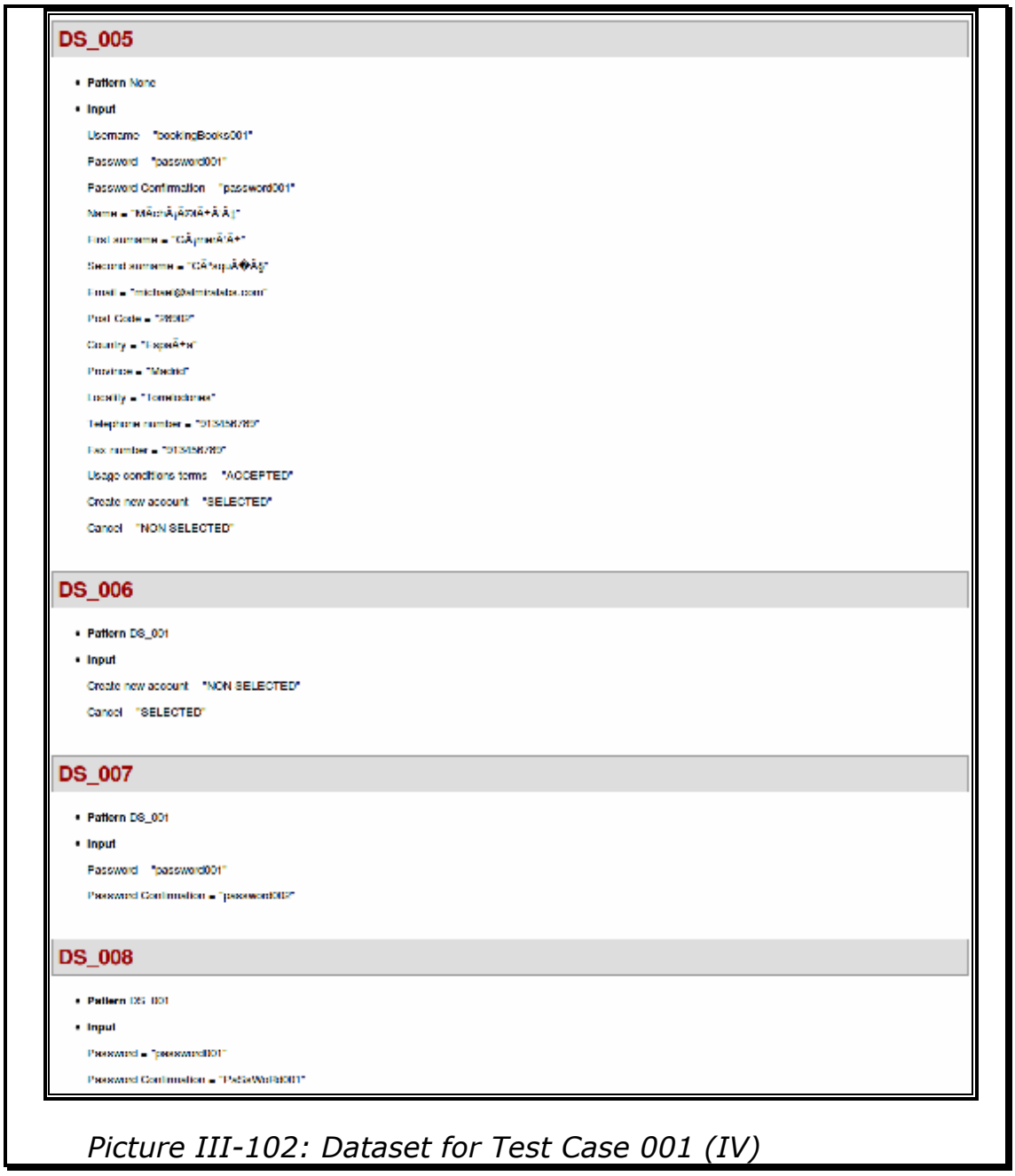
Picture III-99: Dataset for Test Case 001 (I)



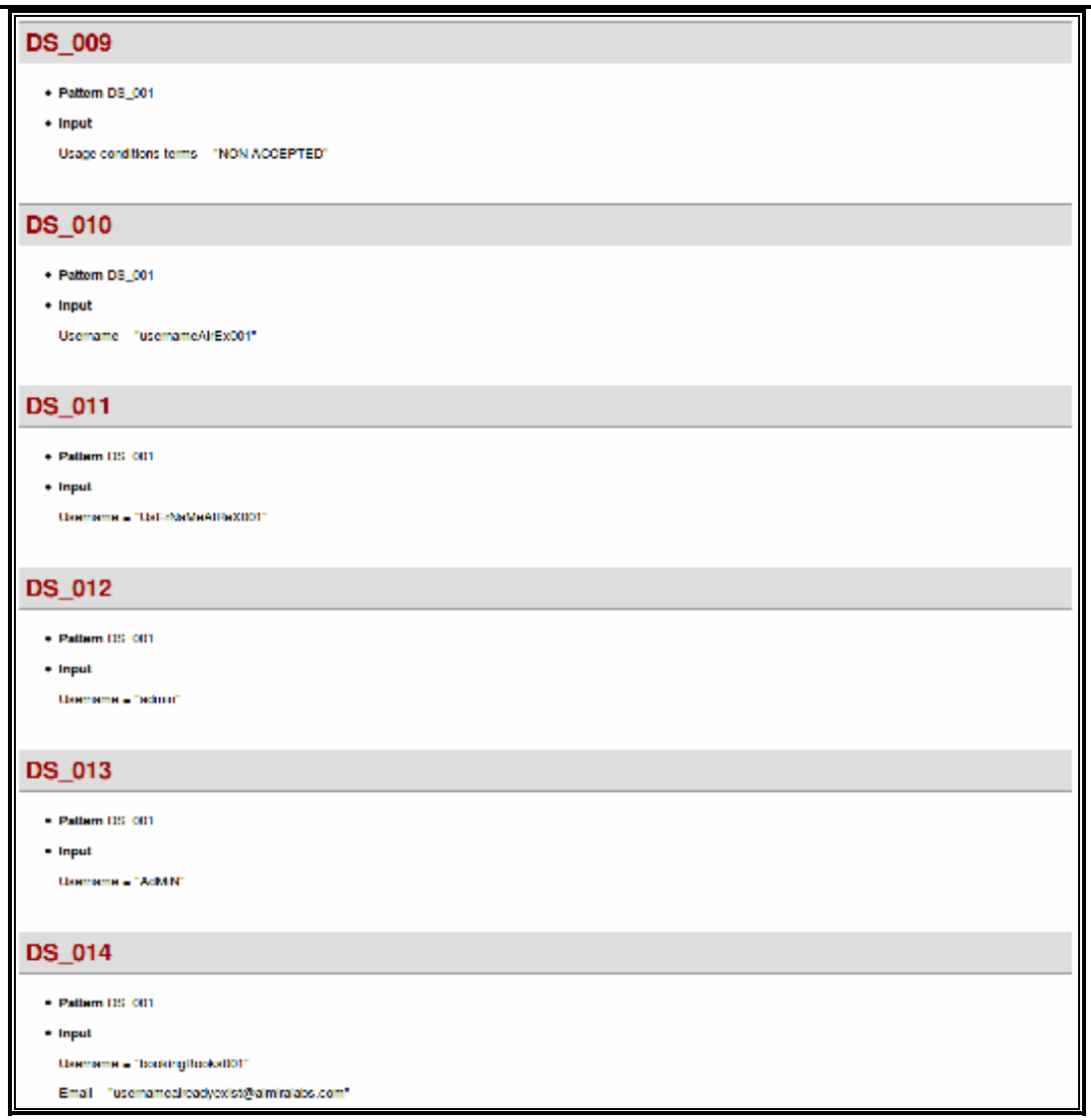
Picture III-100: Dataset for Test Case 001 (II)



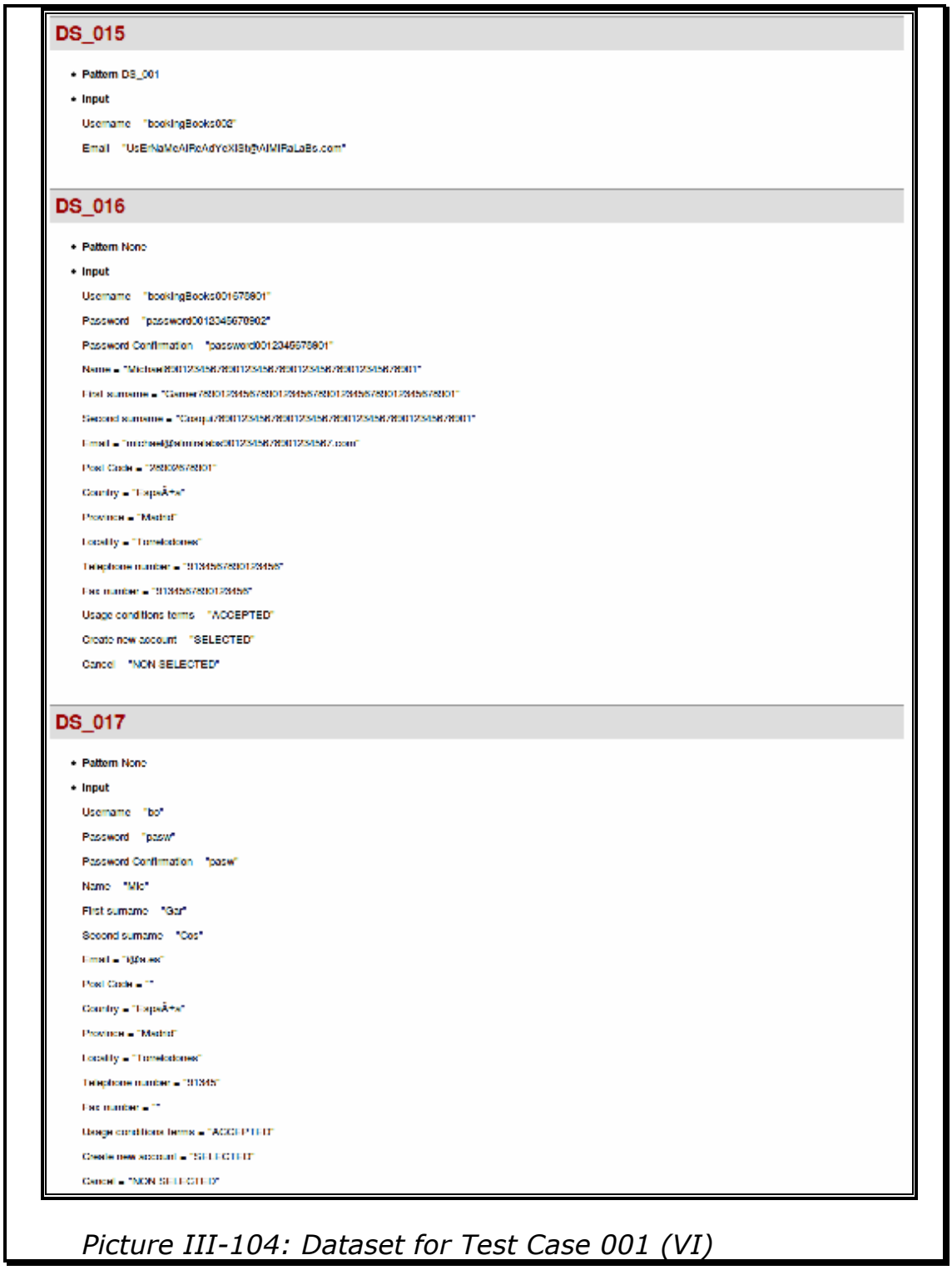
Picture III-101: Dataset for Test Case 001 (III)



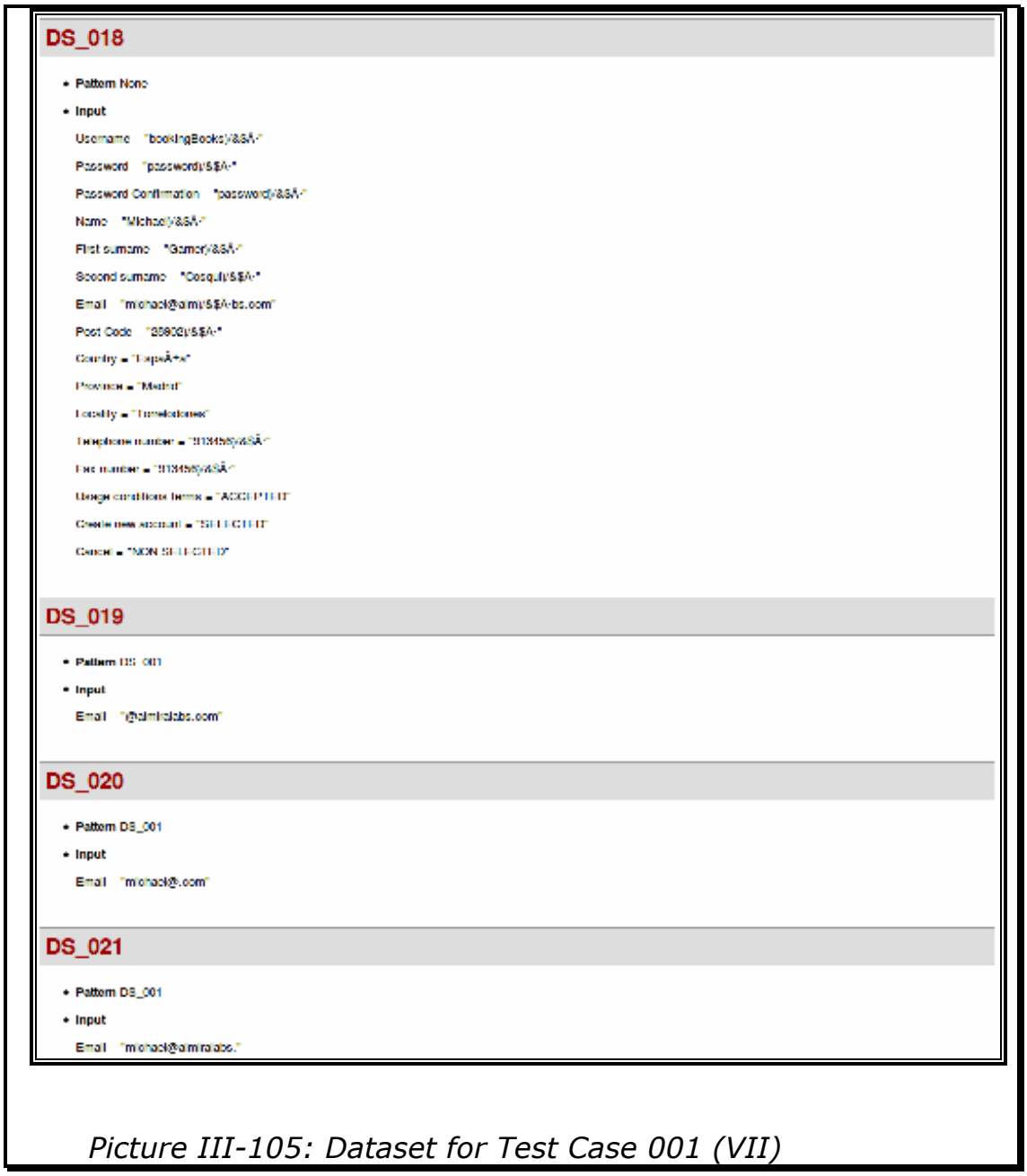
Picture III-102: Dataset for Test Case 001 (IV)



Picture III-103: Dataset for Test Case 001 (V)

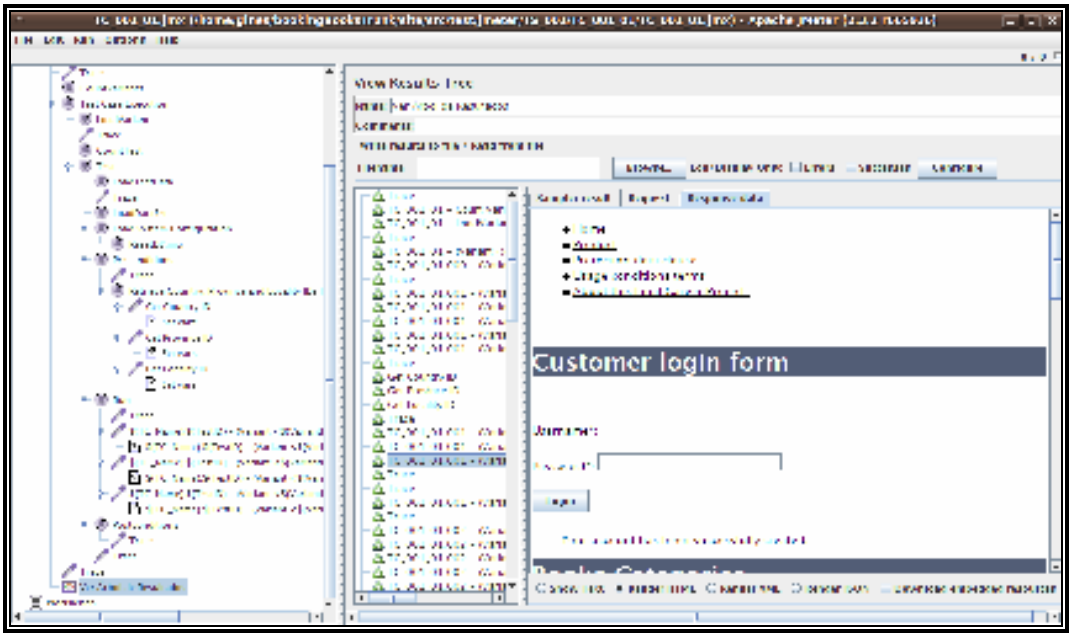


Picture III-104: Dataset for Test Case 001 (VI)

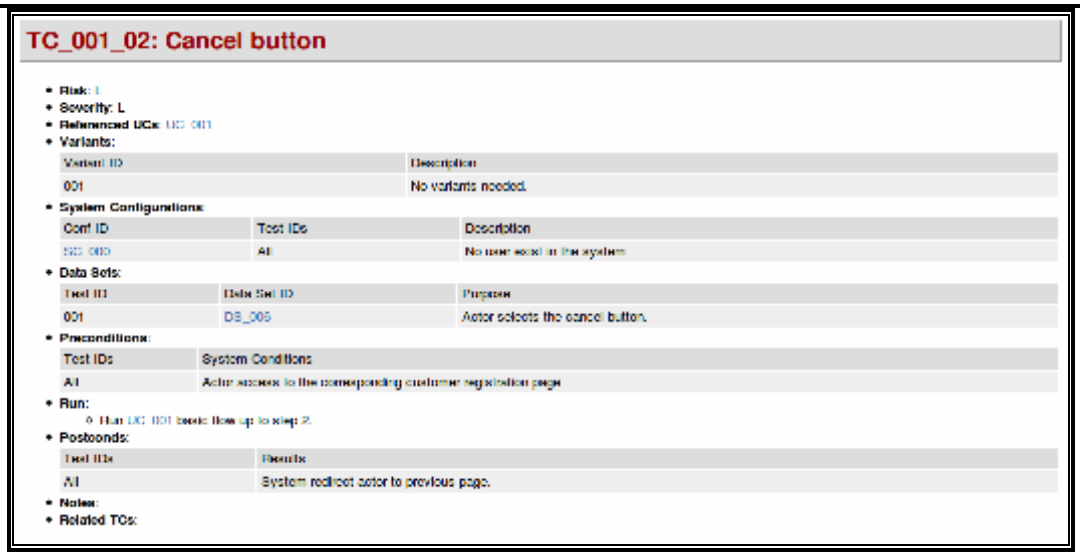


Picture III-105: Dataset for Test Case 001 (VII)

Picture III-106: Dataset for Test Case 001 (VIII)



Picture III-107: Test Case 001_01: Basic Flow – Jmeter Execution



TC_001_02: Cancel button

- **Risk:** L
- **Severity:** L
- **Referenced UCs:** UC_001
- **Variants:**

Variant ID	Description
001	No variants needed.

- **System Configurations:**

Conf ID	Test IDs	Description
SC_000	All	No user exist in the system

- **Data Sets:**

Test ID	Data Set ID	Purpose
001	DS_005	Actor selects the cancel button.

- **Preconditions:**

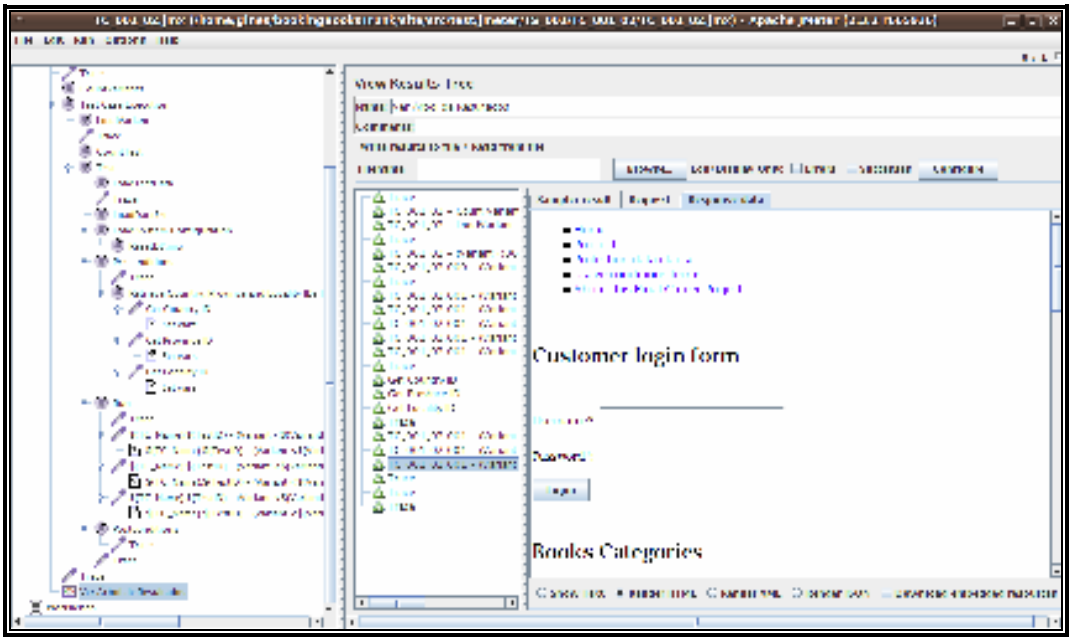
Test IDs	System Conditions
All	Actor access to the corresponding customer registration page

- **Run:**
  1. Run UC_001 basic flow up to step 2.
- **Postconds:**

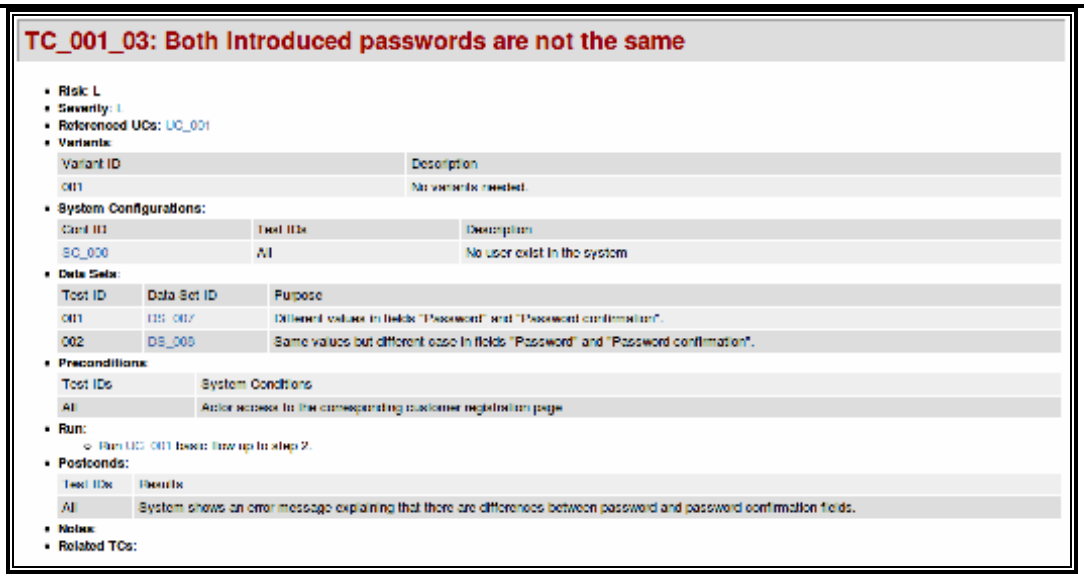
Test IDs	Results
All	System redirect actor to previous page.

- **Notes:**
- **Related TCs:**

Picture III-108: Test Case 001_02: Cancel Button



Picture III-109: Test Case 001_02: Cancel Button – Jmeter Execution



TC_001_03: Both Introduced passwords are not the same

- **Risk:** L
- **Severity:** I
- **Referenced UCs:** UC_001
- **Variants:**

Variant ID	Description
001	No variants needed.

- **System Configurations:**

Conf ID	Test IDs	Description
SC_000	All	No user exist in the system

- **Data Sets:**

Test ID	Data Set ID	Purpose
001	DS_001	Different values in fields "Password" and "Password confirmation".
002	DS_000	Same values but different case in fields "Password" and "Password confirmation".

- **Preconditions:**

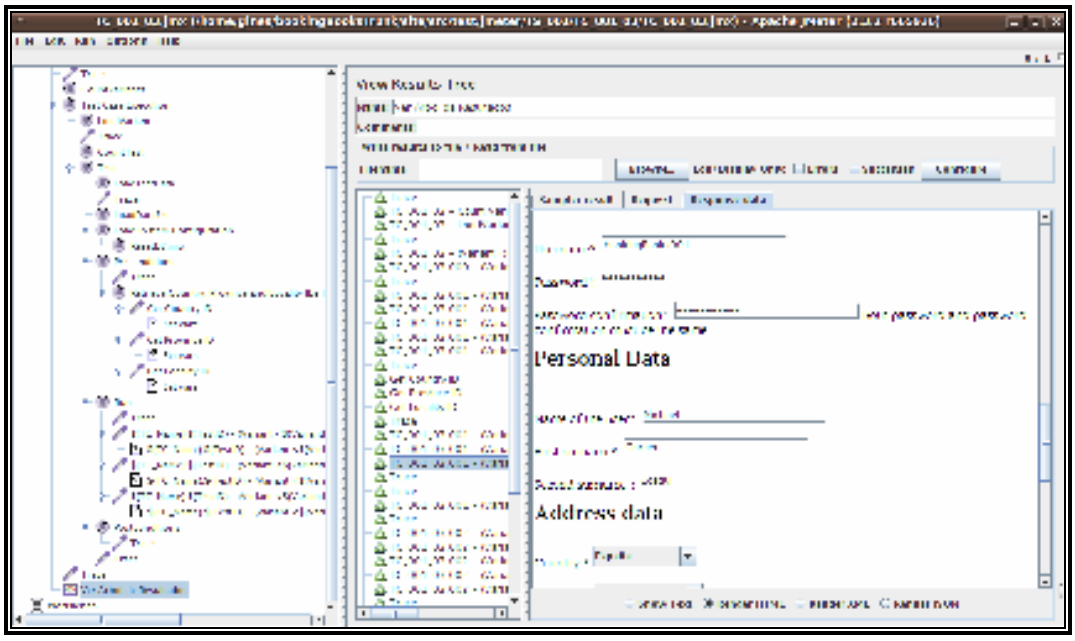
Test IDs	System Conditions
All	Actor access to the corresponding customer registration page

- **Run:**
  - Run UC_001 basic flow up to step 9.
- **Postconds:**

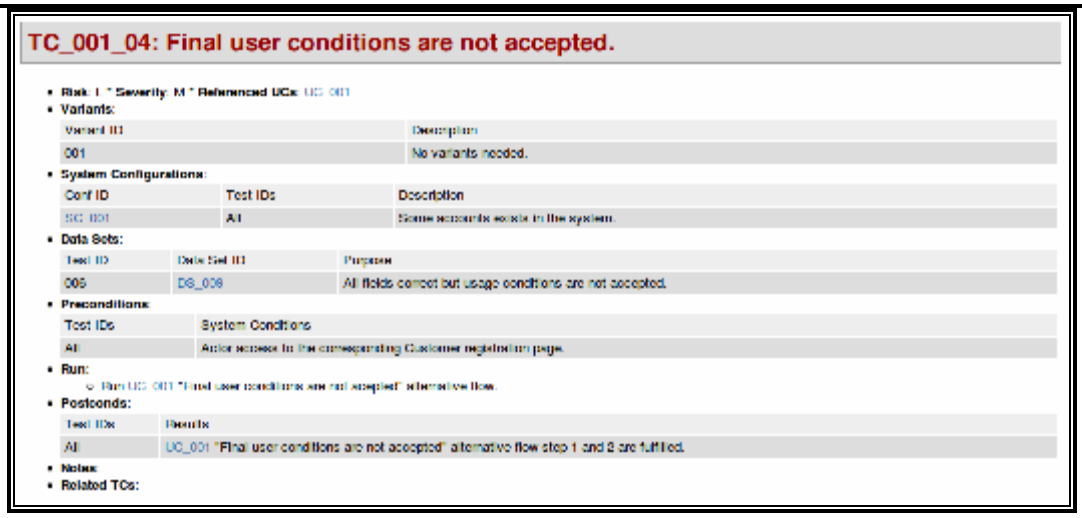
Test IDs	Results
All	System shows an error message explaining that there are differences between password and password confirmation fields.

- **Notes:**
- **Related TCs:**

Picture III-110: TC_001_03: Both Introduced Passwords Are Not the Same



Picture III-111: TC_001_03: Both Introduced Passwords Are Not the Same – Jmeter Execution



TC_001_04: Final user conditions are not accepted.

- **Risk:** L * **Severity:** M * **Referenced UCs:** UC 001
- **Variants:**

Variant ID	Description
001	No variants needed.

- **System Configurations:**

Conf ID	Test IDs	Description
SC 001	All	Some accounts exists in the system.

- **Data Sets:**

Test ID	Data Set ID	Purpose
005	DS_005	All fields correct but usage conditions are not accepted.

- **Preconditions:**

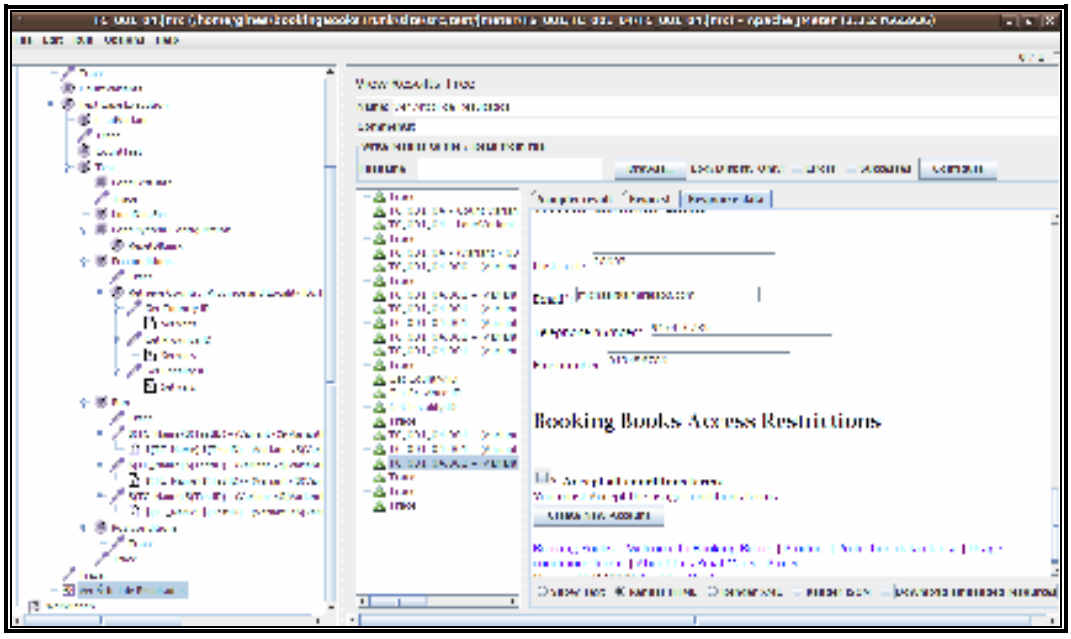
Test IDs	System Conditions
All	Actor access to the corresponding Customer registration page.

- **Run:**
  - Run UC 001 "Final user conditions are not accepted" alternative flow.
- **Postconds:**

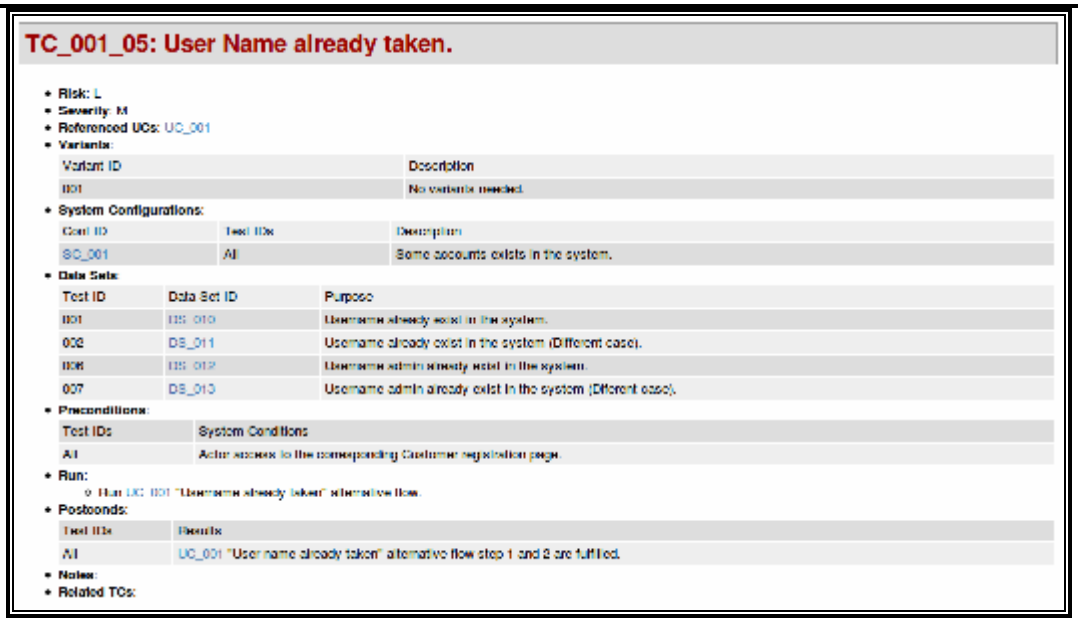
Test IDs	Results
All	UC_001 "Final user conditions are not accepted" alternative flow step 1 and 2 are fulfilled.

- **Notes:**
- **Related TCs:**

Picture III-112: Final User Conditions Are Not Accepted



Picture III-113: Final User Conditions Are Not Accepted – Jmeter Execution



TC_001_05: User Name already taken.

- **Risk:** L
- **Severity:** M
- **Referenced UCs:** UC_001
- **Variants:**

Variant ID	Description
001	No variants needed.

- **System Configurations:**

Conf ID	Test IDs	Description
SC_001	All	Some accounts exists in the system.

- **Data Sets:**

Test ID	Data Set ID	Purpose
001	DS_010	Username already exist in the system.
002	DS_011	Username already exist in the system (Different case).
006	DS_012	Username admin already exist in the system.
007	DS_013	Username admin already exist in the system (Different case).

- **Preconditions:**

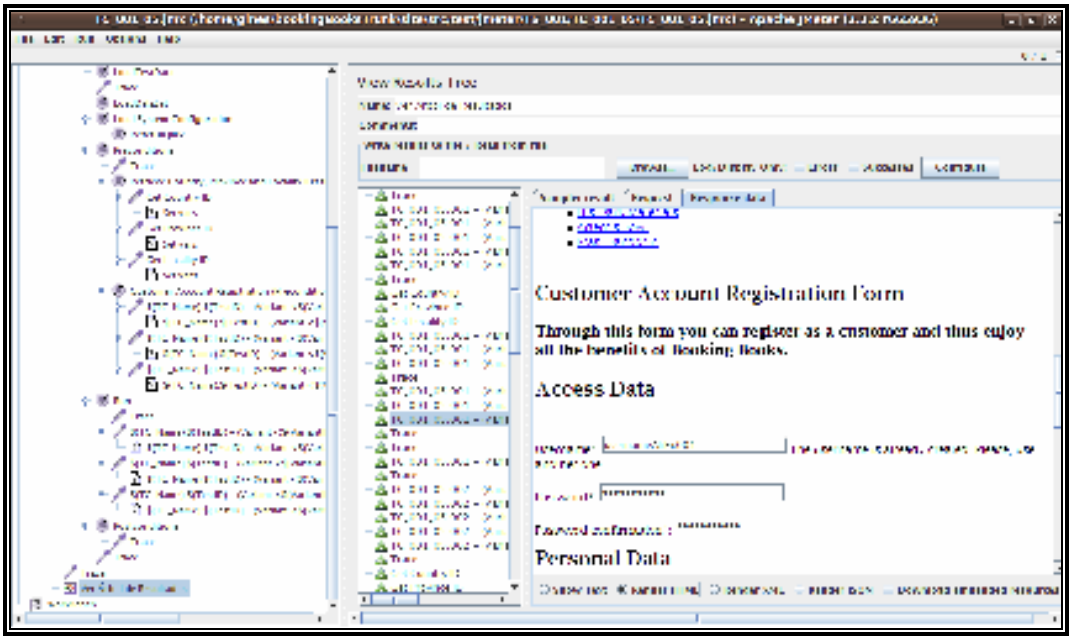
Test IDs	System Conditions
All	Actor access to the corresponding Customer registration page.

- **Run:**
  - Run UC 001 "Username already taken" alternative flow.
- **Postconds:**

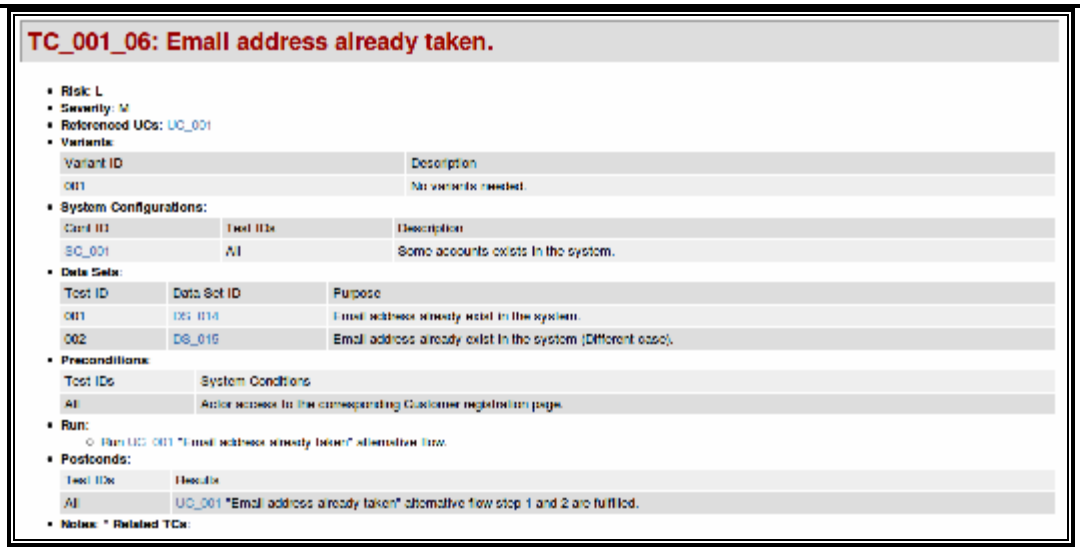
Test IDs	Results
All	UC_001 "User name already taken" alternative flow step 1 and 2 are fulfilled.

- **Notes:**
- **Related TCs:**

Picture III-114: TC_001_05: User Name Already Taken



Picture III-115: TC_001_05: User Name Already Taken – Jmeter Execution



TC_001_06: Email address already taken.

- **Risk:** L
- **Severity:** M
- **Referenced UCs:** UC_001
- **Variants:**

Variant ID	Description
001	No variants needed.

- **System Configurations:**

Conf ID	Test IDs	Description
SC_001	All	Some accounts exists in the system.

- **Data Sets:**

Test ID	Data Set ID	Purpose
001	DS_014	Email address already exist in the system.
002	DS_015	Email address already exist in the system (Different case).

- **Preconditions:**

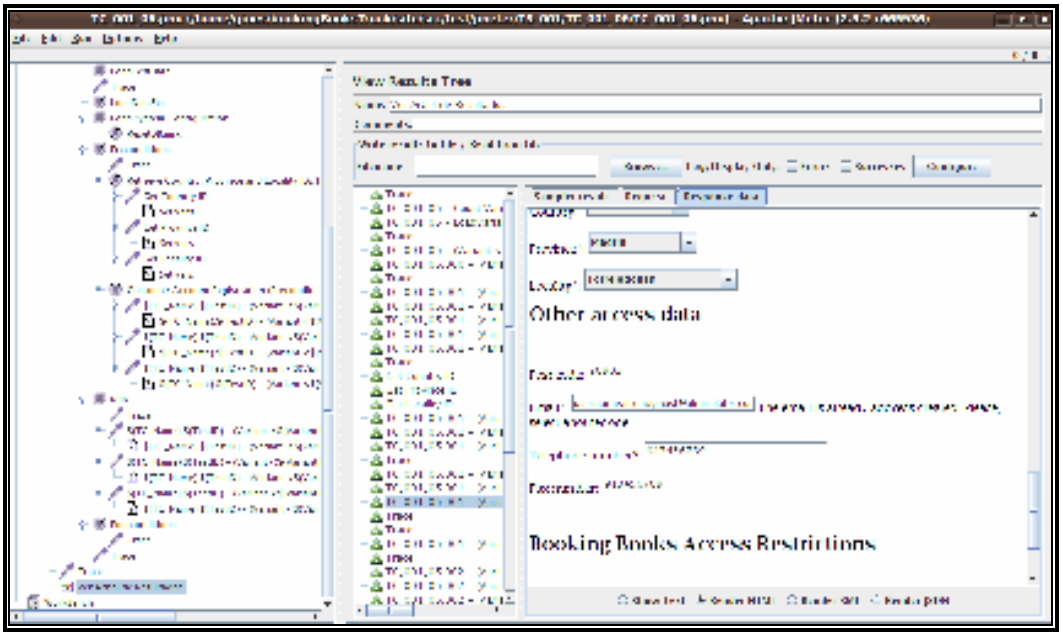
Test IDs	System Conditions
All	Actor access to the corresponding Customer registration page.

- **Run:**
  - Run UC_001 "Email address already taken" alternative flow.
- **Postconds:**

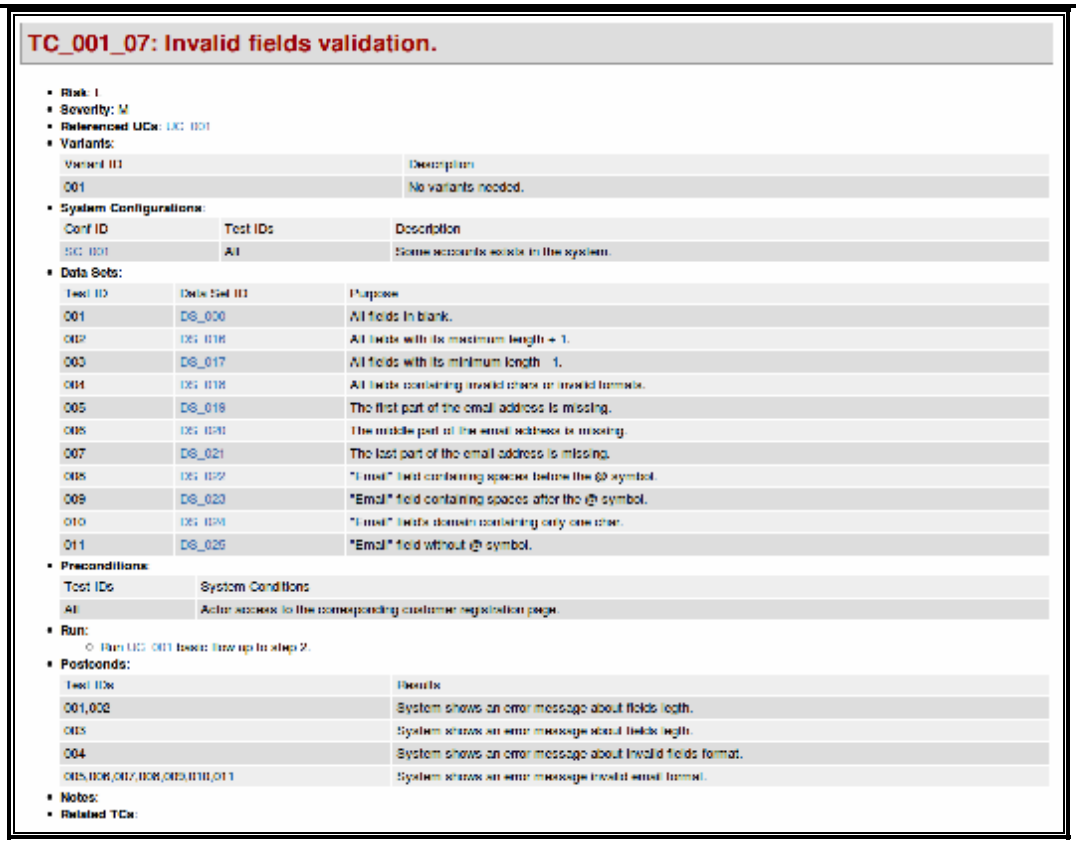
Test IDs	Results
All	UC_001 "Email address already taken" alternative flow step 1 and 2 are fulfilled.

- **Notes:** * Related TCs:

Picture III-116: Email Address Already Taken



Picture III-117: Email Address Already Taken – Jmeter Execution



TC_001_07: Invalid fields validation.

- **Risk:** I
- **Severity:** M
- **Referenced UCs:** UC 001
- **Variants:**

Variant ID	Description
001	No variants needed.

- **System Configurations:**

Conf ID	Test IDs	Description
SC 001	All	Some accounts exists in the system.

- **Data Sets:**

Test ID	Data Set ID	Purpose
001	DS_000	All fields in blank.
002	DS_016	All fields with its maximum length + 1.
003	DS_017	All fields with its minimum length - 1.
004	DS_018	All fields containing invalid chars or invalid formats.
005	DS_019	The first part of the email address is missing.
006	DS_020	The middle part of the email address is missing.
007	DS_021	The last part of the email address is missing.
008	DS_022	"Email" field containing spaces before the @ symbol.
009	DS_023	"Email" field containing spaces after the @ symbol.
010	DS_024	"Email" fields domain containing only one char.
011	DS_025	"Email" field without @ symbol.

- **Preconditions:**

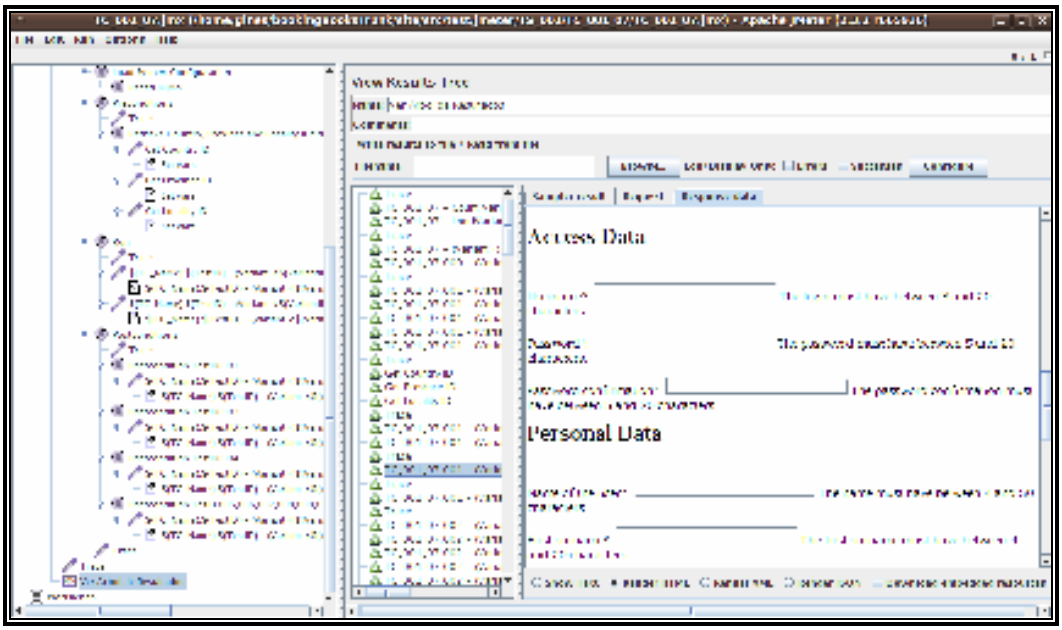
Test IDs	System Conditions
All	Actor access to the corresponding customer registration page.

- **Run:**
  - Run UC 001 basic flow up to step 2.
- **Postconds:**

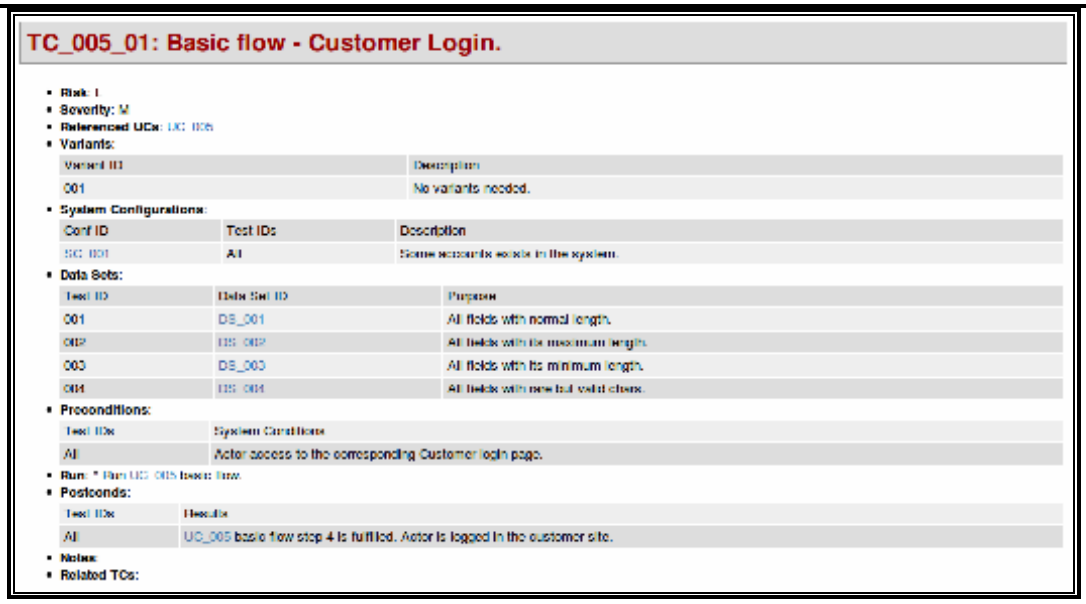
Test IDs	Results
001,002	System shows an error message about fields legth.
003	System shows an error message about fields legth.
004	System shows an error message about invalid fields format.
005,006,007,008,009,010,011	System shows an error message invalid email format.

- **Notes:**
- **Related TCs:**

Picture III-118: TC_001_07: Invalid Fields Validation



Picture III-119: TC_001_07: Invalid Fields Validation – Jmeter Execution



TC_005_01: Basic flow - Customer Login.

- **Risk:** I
- **Severity:** M
- **Referenced UCs:** UC 005
- **Variants:**

Variant ID	Description
001	No variants needed.

- **System Configurations:**

Conf ID	Test IDs	Description
SC 001	All	Some accounts exists in the system.

- **Data Sets:**

Test ID	Data Set ID	Purpose
001	DS_001	All fields with normal length.
002	DS_002	All fields with its maximum length.
003	DS_003	All fields with its minimum length.
004	DS_004	All fields with one but valid chars.

- **Preconditions:**

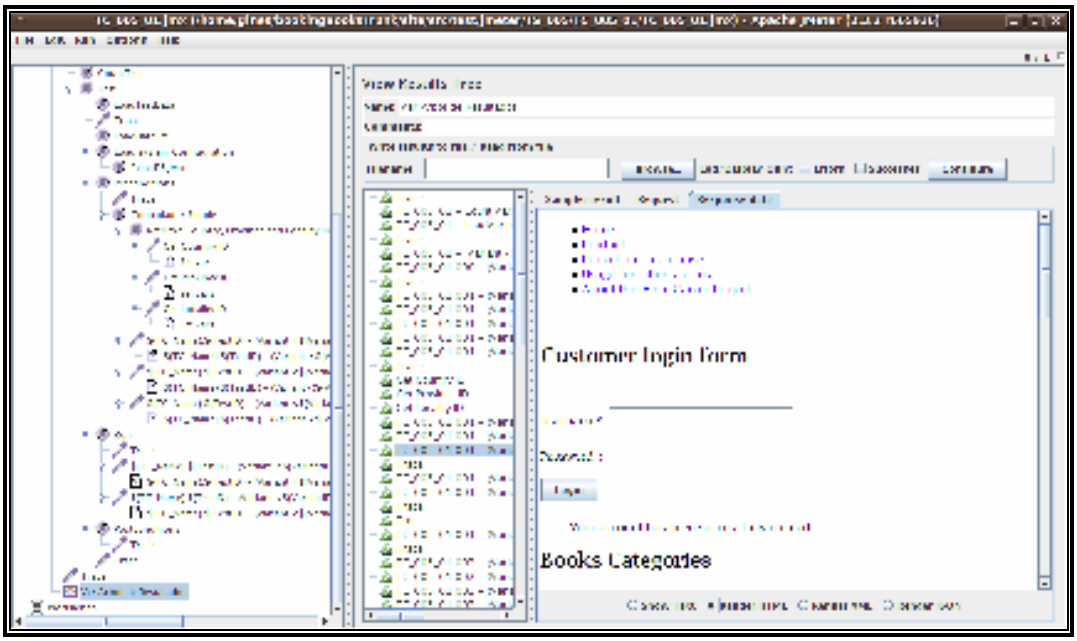
Test IDs	System Conditions
All	Actor access to the corresponding Customer login page.

- **Run:** * Run UC 005 basic flow.
- **Postconds:**

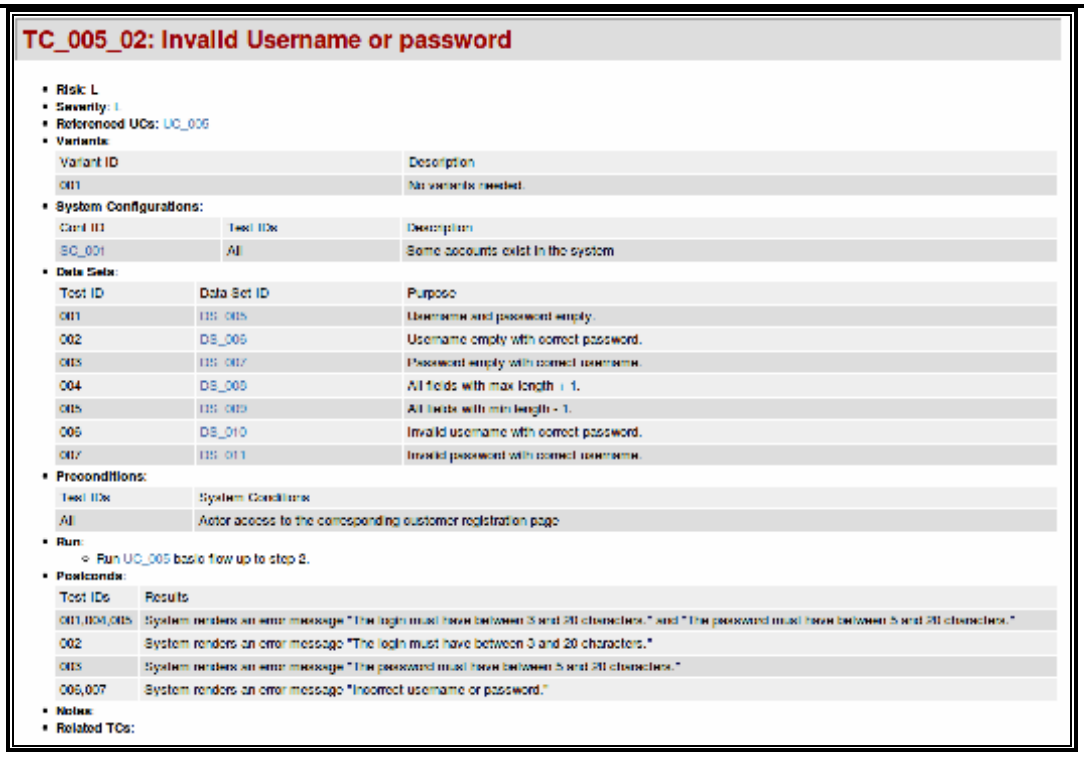
Test IDs	Results
All	UC_005 basic flow step 4 is fulfilled. Actor is logged in the customer site.

- **Notes:**
- **Related TCs:**

Picture III-120: TC_005_01: Basic Flow – Customer Login



Picture III-121: Tc_005_01: Basic Flow – Customer Login – Jmeter Execution



TC_005_02: Invalid Username or password

- **Risk:** L
- **Severity:** I
- **Referenced UCs:** UC_005
- **Variants:**

Variant ID	Description
001	No variants needed.

- **System Configurations:**

Conf ID	Test IDs	Description
SC_001	All	Some accounts exist in the system

- **Data Sets:**

Test ID	Data Set ID	Purpose
001	DS_005	Username and password empty.
002	DS_006	Username empty with correct password.
003	DS_007	Password empty with correct username.
004	DS_008	All fields with max length + 1.
005	DS_009	All fields with min length - 1.
006	DS_010	Invalid username with correct password.
007	DS_011	Invalid password with correct username.

- **Preconditions:**

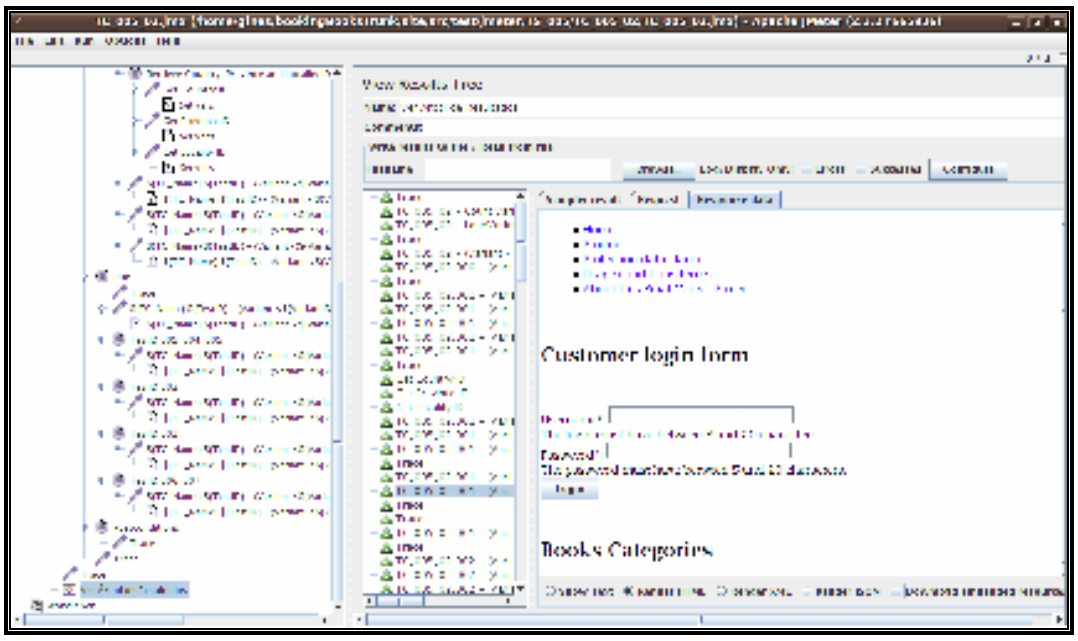
Test IDs	System Conditions
All	Actor access to the corresponding customer registration page

- **Run:**
  - Run UC_005 basic flow up to step 2.
- **Postconds:**

Test IDs	Results
001,004,005	System renders an error message "The login must have between 3 and 20 characters." and "The password must have between 5 and 20 characters."
002	System renders an error message "The login must have between 3 and 20 characters."
003	System renders an error message "The password must have between 5 and 20 characters."
006,007	System renders an error message "Incorrect username or password."

- **Notes:**
- **Related TCs:**

Picture III-122: TC_005_02: Invalid Username or Password



Picture III-123: Tc_005_02: Invalid Username or Password – Jmeter Execution

4. Unit Acceptance Tests

UAT is an acronym for **Unit Acceptance Test**, which confirms that the unit has been installed and that it is possible to communicate with the unit.

This test is a test of the entire architecture including routers, data backup devices, firewalls, etc. This will require the development of a testing protocol that may include testing scripts. The testing should be planned bearing in mind risk issues and contractual obligations.

UAT must be done with the project leader in presence of the web application customer. They must confirm that initial requirements have been developed in the use cases; both must approve the application, signing every use case and requirement.

7.3.7. Control of design and development changes

ISO 9001:2008 Standard

Design and development changes shall be identified and records maintained. The changes shall be reviewed, verified and validated, as appropriate, and approved before implementation. The review of design and development changes shall include evaluation of the effect of the changes on constituent parts and product already delivered.

Records of the results of the review of changes and any necessary actions shall be maintained (see 4.2.4).

ISO 90003:2004 Guideline

In the software development environment, control of design and development changes is usually addressed as part of configuration management (see 7.5.3).

Changes to a software specification or component should maintain appropriate consistency between requirements, designs, code, tests specifications, user manuals and, where relevant, other additional items.

NOTE 1 For further information see ISO/IEC 12207:1995[11], 5.5.2, 5.5.3 (modifications), 6.1 and 6.2 (configuration management), and ISO/IEC 12207:1995/Amd.1:2002[12], F.2.1 (documentation) and F.2.2 (configuration management).

NOTE 2 For further general guidance related to ISO 9001:2000, 7.3, see the following:

— ISO/IEC 12207:1995/Amd.1:2002[12], F.1.3.4 (software requirements analysis) and F.1.3.5 (software design);

— ISO/IEC 12119:1994[10] for guidance on any procured COTS software products;

— ISO/IEC 6592:2000[1] for design and development documentation guidance;

— ISO/IEC 19761[31], ISO/IEC 20926[32] and ISO/IEC 20968[33] for guidance on estimation of size methods;

— ISO/IEC TR 14759[18] for guidance on prototype categorization and examples of use;

— ISO/IEC 15910[28] for software user documentation process.

Guideline Application

1. Description

Stable designs are very rare. Most designs are subjected to frequent changes, sometimes even before the design process is finished. It is as important to control these changes as it is to control the original design and development process. It should be clear how these changes are handled and what effects they have on the product.

Is necessary to handle any changes to designs similar to the way new designs are handled. When changes to design and development are needed, the design and development plan must incorporate these changes.

Guideline Application Tool

1. Description

Changes will be done opening an issue of type *Change Request* in *Jira*.

7.4. Purchasing

7.4.1. Purchasing process

ISO 9001:2008 Standard

The organization shall ensure that purchased product conforms to specified purchase requirements. The type and extent of control applied to the supplier and the purchased product shall be dependent upon the effect of the purchased product on subsequent product realization or the final product.

The organization shall evaluate and select suppliers based on their ability to supply product in accordance with the organization's requirements. Criteria for selection, evaluation and re-evaluation shall be established. Records of the results of evaluations and any necessary actions arising from the evaluation shall be maintained (see 4.2.4).

Guideline Application

1. Description

For the purchase process, I have created a general management purchase template in section *4.2.1.27. Purchase Management Template*, which will be a guideline for all entire section 7.4.

2. Purchase Management

2.1. Objective

Ensure the acquisition of property under and / or contracting services through the definition of guidelines for compliance and improvement of conditions of quality, price and delivery times

2.2. Scope

This process applies to the purchase of goods and services critical to the events of tradition and cultural programs, classified into the following categories: Technical, Logistics and Support Services, Consulting, General Services Administration and Promotion. In addition the control of suppliers who supply the products and services. The procedure is structured under the methodology of continuous improvement cycle PDCA (Plan, Do or Run as planned, Check and Act or improve the process activities and their results).

Guideline Application Tool

1. Description

Purchase Management Template must be published in *Twiki*.

7.4.1.1. Purchased products

ISO 90003:2004 Guideline

For the purposes of 7.4.1, free software (such as open source development tools) should be considered as purchased.

In developing, supplying, installing and maintaining software products, types of purchased products may include:

a) COTS software or shareware;

b) customized software and services;

c) subcontracted development (e.g. contract staff or outsourced full product development);

d) outsourced activities (e.g. testing, independent verification and validation, facilities management);

e) tools intended to assist in the development of software (e.g. design and development or configuration management tools, code analyzers, debuggers, test analyzers, generators, compilers);

f) computer and communications hardware;

g) key components (e.g. integrated circuits may be subject to change or to uncertain continued availability);

h) user and product documentation;

i) training courses and materials.

The type and extent of control to be exercised by the organization over a supplier of subcontracted design or development (e.g. joint projects) becomes especially important when selecting the supplier, because confidence in the relationship may be critical to the success of the development.

In developing, supplying, installing and maintaining software products, consideration about purchased products may require the organization to manage the risks associated with licensing, maintenance, help desk, and customer support services (such as concern for continued availability of support for purchased product as a result of later releases). One way of determining the capability of suppliers to provide an acceptable product may be by performing process assessment. Process assessment provides information for risk assessment and a view of maturity and capability level of the supplier's processes.

7.4.1.2. Purchased product control

ISO 90003:2004 Guideline

Where the products listed in 7.4.1.1 a) to i) are purchased and intended to become part of the product, they should be controlled as components throughout the design and development. Contractual considerations should be addressed to ensure that such controls are in place to ensure configuration management is effective.

Care should be taken to ensure that contract staff has the specific skills and the levels of competence required prior to being integrated as part of the project team.

Re-evaluation of suppliers' performance may be conducted by regular review and control during design and development as part of project management (see 7.3.1).

In some circumstances, the whole of ISO 9001:2008 may apply to the organization–supplier relationship. The management of risk is often more critical in software development because of the nature of the product.

The supplier may be selected based upon the evaluation of the supplier's proposals and process capabilities, and other factors, such as analysis of a supplier's performance history, review of the responses to the supplier questionnaire, and review of software-related quality and verification plans.

NOTE 1 For further information, see ISO/IEC 12207:1995[11], 5.1 (acquisition process), and ISO/IEC 12207:1995/Amd.1:2002[12], F.1.1 (acquisition process).

NOTE 2 For further information on assessing process capability of a supplier, see ISO/IEC 15504-3[24].

7.4.2. Purchasing information

ISO 9001:2008 Standard

Purchasing information shall describe the product to be purchased, including, where appropriate,

a) requirements for approval of product, procedures, processes and equipment,

b) requirements for qualification of personnel, and

c) quality management system requirements.

The organization shall ensure the adequacy of specified purchase requirements prior to their communication to the supplier.

ISO 90003:2004 Guideline

Purchasing information for software may include, where applicable

a) identification of the product ordered (such as product name, number, version, configuration),

b) requirements or the procedure to identify requirements where not fixed at the time of order,

c) standards to be applied (e.g. communications protocol, architectural specification, coding standards),

d) procedures and/or work instructions the supplier is instructed to follow,

e) description of the development environment (e.g. hardware, development tools, facilities),

f) description of the target environment (e.g. hardware, operating system), and

g) requirements on personnel (e.g. prerequisite training, product knowledge).

The considerations covered in 7.2.2 may also be applied to subcontracts.

NOTE For further information, see ISO/IEC 12207:1995[11], 5.1.2 [request-for-proposal (-tender) preparation], and ISO/IEC 12207:1995/Amd.1:2002[12], F.1.1.1 (acquisition preparation).

7.4.3. Verification of purchased product

ISO 9001:2008 Standard

The organization shall establish and implement the inspection or other activities necessary for ensuring that purchased product meets specified purchase requirements.

Where the organization or its customer intends to perform verification at the supplier's premises, the organization shall state the intended verification arrangements and method of product release in the purchasing information.

ISO 90003:2004 Guideline

This verification can apply to the acceptance testing of purchased software used in development. Much of such software is impossible to verify thoroughly because of its extensive functionality. The organization is entitled to assume a degree of suitability but should conduct acceptance testing and ensure the availability of adequate support.

Where part of the software development has been subcontracted, or where the purchase of associated hardware and software is involved, the organization may need to determine the methods by which verification, validation and acceptance of the subcontracted work will be achieved. Where software developed under subcontract has to be integrated with software developed by the organization itself, other considerations may include the methods and tools used in the development. Inspection by

the organization itself, and possibly by the customer, may be necessary. The general considerations for testing apply (see 8.2.4).

The organization may be required to acquire and include software products, including data or services such as contract staff, supplied by a third party. The organization should verify product and services upon receipt, taking into account the requirements of the contract. The methods to verify the product may need to be defined as part of the purchasing requirements (such as acceptance testing). The guidance on verification and validation provided in 7.3.5 and 7.3.6 should be considered. For contract staffing, consideration should be given to the education, training, skills and experience of such staff, in such topics as programming language, development tools and system management.

When purchasing or obtaining data, careful consideration should be given to the format, medium, volume, source and content of data obtained (e.g. test data obtained from a third party). Data protection regulatory requirements may be relevant in some cases (e.g. privacy).

When purchasing software products, consideration should be given to the format and medium on which it is supplied to ensure operational requirements have been met. The functional and performance requirements of the product should be tested before use to ensure that the product performs as specified. The product may also need to be validated against the needs of the final product it is required to satisfy.

Since it may not always be possible to test the product at the point of receipt, it is important to ensure that it is tested before use or incorporation into the final product. Such tests may need to be conducted at the supplier’s premises. When on the organization's premises, consideration should be given to ensuring that appropriate measures are in place to segregate the product until certainty about its integrity can be determined (e.g. virus infections).

Records of qualification and training records may be used to assist with verification of contract staff.

NOTE 1 For further information, see ISO/IEC 12207:1995[11], 5.1.5, and ISO/IEC 12207:1995/Amd.1:2002[12], F.1.1.4 (acquisition — customer acceptance).

NOTE 2 For further general guidance related to ISO 9001:2000, 7.4, see the following:

— ISO/IEC 9126-1:2001[5] for guidance on quality characteristics appropriate to purchasing software product;

— ISO/IEC 14598-4[16];

— ISO/IEC 19761[31], ISO/IEC 20926[32] and ISO/IEC 20968[33] for guidance on estimation of size methods.

7.5. Product and service provision

7.5.1. Control of production and service provision

ISO 9001:2008 Standard

The organization shall plan and carry out production and service provision under controlled conditions. Controlled conditions shall include, as applicable

a) the availability of information that describes the characteristics of the product,

b) the availability of work instructions, as necessary,

c) the use of suitable equipment,

d) the availability and use of monitoring and measuring equipment,

e) the implementation of monitoring and measurement, and

f) the implementation of release, delivery and post-delivery activities.

7.5.1.1. Production and service provision in software

ISO 90003:2004 Guideline

As stated in the guidance for design and development (see 7.3), a software development project should be organized according to a set of processes, which transform the requirements into a software product. The *control of production and service provision* requirements specified in ISO 9001:2008, 7.5.1 is equivalent for software products to

a) release activities, e.g. build, release, and replication,

b) delivery activities, e.g. delivery and installation, and

c) post-delivery activities, e.g. operations, maintenance and customer support (these apply throughout the life of the product).

Guideline Application

1. Description

Release activities, delivery activities and post-delivery activities are described in the next sections.

Guideline Application Tool

1. Description

It is not necessary add any guideline application tool.

7.5.1.2. Build and release

ISO 90003:2004 Guideline

Processes should be set up for build, release and replication of the software item(s). Build and release invoke configuration management (see 7.5.3, identification and traceability).

The following provisions are appropriate to build and release:

a) identification of the software items that constitute each release, including associated build instructions;

b) identification of the types (or classes) of release, depending on the frequency and/or impact on the customer's operations and ability to implement changes at any point in time;

c) decision criteria and guidance to determine where localized temporary fixes may be incorporated or release of a complete updated copy of the software product is necessary.

Guideline Application

1. Description

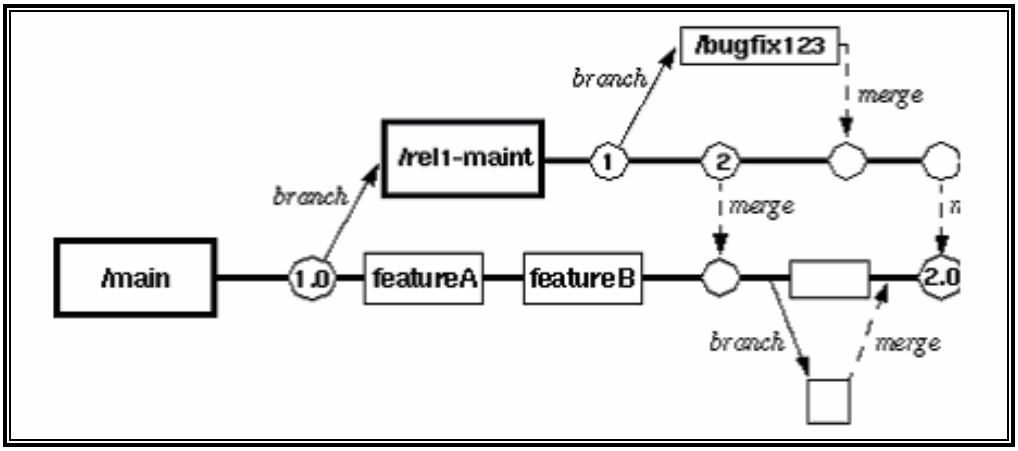
The release process can be described with the next three concepts:

- **Trunk**: Main development area. This is where your next major release of the code lives, and generally has all the newest features. Could be the main body of development, originating from the start of the project until the present. *Booking Books* is on the version *1.3.4 SNAPSHOT*.

- **Tag**: Every time you release a version (according to our USDP software life cycle, a release will be created every one, two or three weeks, and they could be) you make a tag for it. This gives you a point-in-time copy of the code as it was at that state, allowing you to go back and reproduce any bugs if necessary in a past version, or re-release a past version exactly as it was. Branches and tags in SVN are lightweight - on the server, it does not make a full copy of the files, just a marker saying "these files were copied at this revision" that only takes up a few bytes. With this in mind, you should never be concerned about creating a tag for any released code. Once the tag is created, you should never add changes or modify code on it. When done, *Booking Books*

version *1.3.4 SNAPSHOT* will be *Booking Books* version *1.3.4* (and will be created the version *1.3.5 SNAPSHOT* for future releases).

• **Branch**: Every time you release a major version, it gets a branch created. This allows you to do bug fixes and make a new release without having to release the newest - possibly unfinished or untested - features. Brachs are used when are necessary a parallel development (for example, adding new software features and fixing bugs by QA).



Picture III-124: Release Sample Version Tree Diagram

Booking Books started with versions 1.0.0; now it is in the version 1.3.4 SNAPSHOT

Guideline Application Tool

1. Description

In this Project I am using *Maven* as build and release tool.

Maven is based around the central concept of a build lifecycle. What this means is that the process for building and distributing a particular artifact (project) is clearly defined.

For the person building a project, this means that it is only necessary to learn a small set of commands to build any Maven project, and the POM (the Project Object Model, an XML file that contains information about the project and configuration details used by *Maven* to build the project) will ensure they get the results they desired.

There are three built-in build lifecycles: default, clean and site. The default lifecycle handles your project deployment, the clean lifecycle handles project cleaning, while the site lifecycle handles the creation of your project's site documentation.

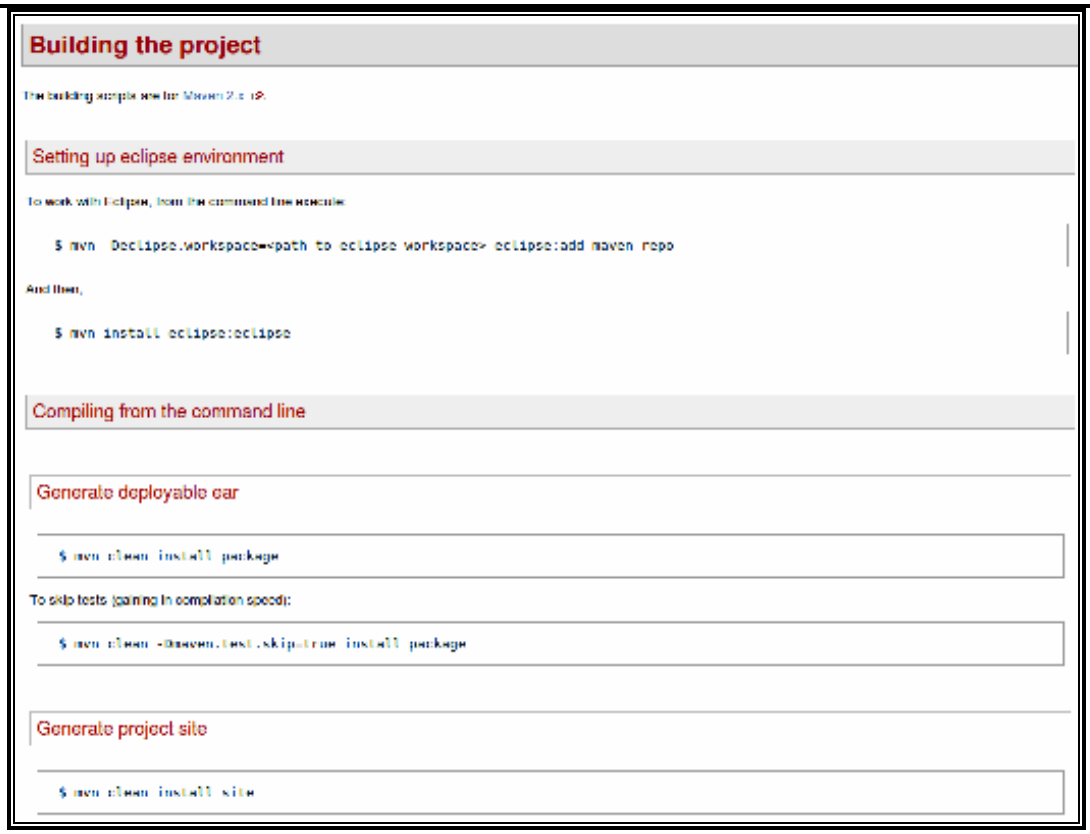
2. Build Life Cycle Phases

Each of these build lifecycles is defined by a different list of build phases, wherein a build phase represents a stage in the lifecycle.

For example, the default lifecycle has the following build phases:

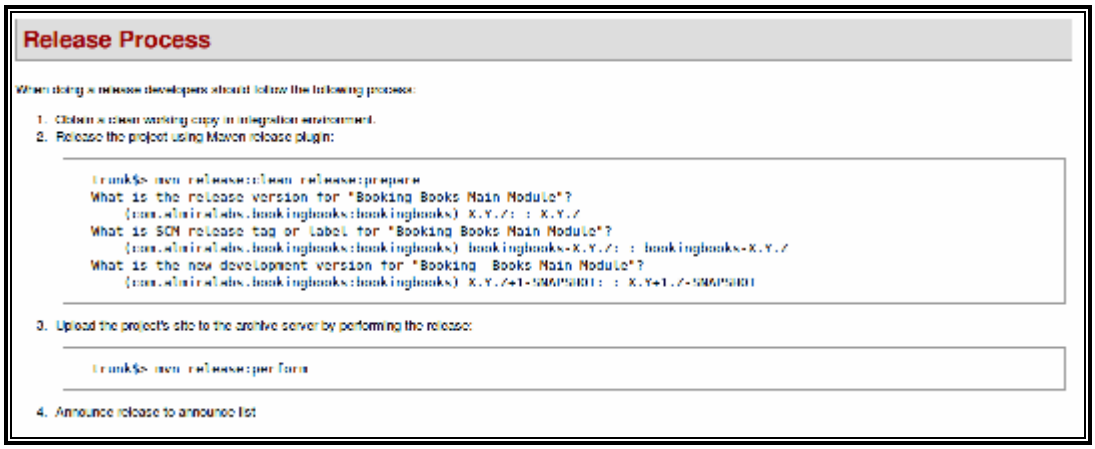
- **Validate:** validate the project is correct and all necessary information is available
- **Compile:** compile the source code of the project
- **Test:** test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed
- **Package:** take the compiled code and package it in its distributable format, such as a JAR.
- **Integration test:** process and deploy the package if necessary into an environment where integration tests can be run
- **Verify:** run any checks to verify the package is valid and meets quality criteria
- **Install:** install the package into the local repository, for use as a dependency in other projects locally
- **Deploy:** done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects.

These build phases (plus the other build phases not shown here) are executed sequentially to complete the default lifecycle. Given the build phases above, this means that when the default lifecycle is used, Maven will first validate the project, then will try to compile the sources, run those against the tests, package the binaries (e.g. jar), run integration tests against that package, verify the package, install the verified package to the local repository, then deploy the installed package in a specified environment.



Picture III-125: Building The Project

- Release process



Picture III-126: Release Process

2. Build Command-Line Example

In this example, I executed the command *mvn -Dmaven.test.skip=true clean install*

gines@skynet::/home/gines/bookingBooksTrunk$mvn -Dmaven.test.skip=true clean install

[INFO] Scanning for projects...

```
[INFO] Reactor build order:
[INFO]   Booking Books Core Module
[INFO]   Booking Books Site Module
[INFO]   Booking Books EAR Module
[INFO]   Booking Books Main Module
[INFO] ------------------------------------------------------------------------
[INFO] Building Booking Books Core Module
[INFO]    task-segment: [clean, install]
[INFO] ------------------------------------------------------------------------
[INFO] [clean:clean {execution: default-clean}]
[INFO] Deleting file set: /home/gines/bookingBooksTrunk/core/target (included:
[**], excluded: [])
[INFO] [resources:resources {execution: default-resources}]
[WARNING] File encoding has not been set, using platform encoding UTF-8, i.e.
build is platform dependent!
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources,
i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory
/home/gines/bookingBooksTrunk/core/src/main/resources
[INFO] Copying 9 resources to .
[INFO] [compiler:compile {execution: default-compile}]
Compiling 54 source files to /home/gines/bookingBooksTrunk/core/target/classes
[INFO] [resources:testResources {execution: default-testResources}]
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources,
i.e. build is platform dependent!
[INFO] Copying 1 resource to .
[INFO] [compiler:testCompile {execution: default-testCompile}]
[INFO] Not compiling test sources
[INFO] [surefire:test {execution: default-test}]
[INFO] Tests are skipped.
[INFO] [jar:jar {execution: default-jar}]
[INFO] Building jar: /home/gines/bookingBooksTrunk/core/target/bookingbooks-
core-1.3.4.jar
[INFO] [install:install {execution: default-install}]
[INFO] Installing /home/gines/bookingBooksTrunk/core/target/bookingbooks-core-
1.3.4.jar to /home/gines/.m2/repository/com/almiralabs/petstore/bookingbooks-
core/1.3.4/bookingbooks-core-1.3.4.jar
[INFO] ------------------------------------------------------------------------
[INFO] Building Booking Books Site Module
[INFO]    task-segment: [clean, install]
[INFO] ------------------------------------------------------------------------
[INFO] [clean:clean {execution: default-clean}]
[INFO] Deleting file set: /home/gines/bookingBooksTrunk/site/target (included:
[**], excluded: [])
[INFO] [resources:resources {execution: default-resources}]
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources,
i.e. build is platform dependent!
[INFO] Copying 2 resources to .
[INFO] skip non existing resourceDirectory
/home/gines/bookingBooksTrunk/site/src/main/config
[INFO] [compiler:compile {execution: default-compile}]
Compiling 11 source files to /home/gines/bookingBooksTrunk/site/target/classes
[INFO] [resources:testResources {execution: default-testResources}]
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources,
i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory
/home/gines/bookingBooksTrunk/site/src/test/groovy
[INFO] Copying 96 resources to .
[INFO] [compiler:testCompile {execution: default-testCompile}]
[INFO] Not compiling test sources
[INFO] [surefire:test {execution: default-test}]
[INFO] Tests are skipped.
[INFO] [war:war {execution: default-war}]
```

```
[INFO] Packaging webapp
[INFO] Assembling webapp[bookingbooks-site] in [/home/gines/bookingBooksTrunk/site/target/bookingbooks-site-1.3.4]
[INFO] Processing war project
[INFO] Copying webapp resources[/home/gines/bookingBooksTrunk/site/src/main/webapp]
[INFO] Webapp assembled in [1562 msecs]
[INFO] Building war: /home/gines/bookingBooksTrunk/site/target/bookingbooks-site-1.3.4.war
[INFO] [install:install {execution: default-install}]
[INFO] Installing /home/gines/bookingBooksTrunk/site/target/bookingbooks-site-1.3.4.war to /home/gines/.m2/repository/com/almiralabs/petstore/bookingbooks-site/1.3.4/bookingbooks-site-1.3.4.war
[INFO] ------------------------------------------------------------------------
[INFO] Building Booking Books EAR Module
[INFO]   task-segment: [clean, install]
[INFO] ------------------------------------------------------------------------
[INFO] [clean:clean {execution: default-clean}]
[INFO] Deleting file set: /home/gines/bookingBooksTrunk/ear/target (included: [**], excluded: [])
[INFO] [ear:generate-application-xml {execution: default-generate-application-xml}]
[INFO] Generating application.xml
[INFO] [resources:resources {execution: default-resources}]
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory /home/gines/bookingBooksTrunk/ear/src/main/resources
[INFO] [ear:ear {execution: default-ear}]
[INFO] Copying artifact[war:com.almiralabs.petstore:bookingbooks-site:1.3.4] to[site.war]
[INFO] Copy ear sources to /home/gines/bookingBooksTrunk/ear/target/bookingbooks-ear-1.3.4
[INFO] Could not find manifest file: /home/gines/bookingBooksTrunk/ear/src/main/application/META-INF/MANIFEST.MF - Generating one
[INFO] Building jar: /home/gines/bookingBooksTrunk/ear/target/bookingbooks-ear-1.3.4.ear
[INFO] [install:install {execution: default-install}]
[INFO] Installing /home/gines/bookingBooksTrunk/ear/target/bookingbooks-ear-1.3.4.ear to /home/gines/.m2/repository/com/almiralabs/petstore/bookingbooks-ear/1.3.4/bookingbooks-ear-1.3.4.ear
[INFO] ------------------------------------------------------------------------
[INFO] Building Booking Books Main Module
[INFO]   task-segment: [clean, install]
[INFO] ------------------------------------------------------------------------
[INFO] [clean:clean {execution: default-clean}]
[INFO] Deleting file set: /home/gines/bookingBooksTrunk/target (included: [**], excluded: [])
[INFO] [site:attach-descriptor {execution: default-attach-descriptor}]
[INFO] Unable to load parent project from a relative path: Could not find the model file '/home/gines/pom.xml'. for project unknown
[INFO] Parent project loaded from repository.
[INFO] [install:install {execution: default-install}]
[INFO] Installing /home/gines/bookingBooksTrunk/pom.xml to /home/gines/.m2/repository/com/almiralabs/bookingbooks/1.3.4/bookingbooks-1.3.4.pom
[INFO]
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO] ------------------------------------------------------------------------
[INFO] Booking Books Core Module ..... SUCCESS [10.990s]
```

```
[INFO] Booking Books Site Module ..... SUCCESS [11.659s]
[INFO] Booking Books EAR Module ..... SUCCESS [4.100s]
[INFO] Booking Books Main Module ..... SUCCESS [4.292s]
[INFO] ------------------------------------------------------------------------
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 33 seconds
[INFO] Finished at: Fri Apr 02 20:22:14 CEST 2010
[INFO] Final Memory: 34M/95M
[INFO] ------------------------------------------------------------------------
```

2. Maven Project Object Model (POM.xml)

2.1 Main POM.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <parent>
      <groupId>com.almiralabs</groupId>
      <artifactId>almira-reports-pom</artifactId>
      <version>1.1.3</version>
   </parent>
   <artifactId>bookingbooks</artifactId>
   <packaging>pom</packaging>
   <name>Booking Books Main Module</name>
   <url>https://devel.almiralabs.com/nightly/petstore/</url>

   <!--Version of this project-->
   <version>1.3.4</version>
   <inceptionYear>2008</inceptionYear>

   <organization>
      <name>Almira Labs</name>
      <url>http://almiralabs.com/</url>
   </organization>

   <!--Parts which compose this project-->
   <modules>
      <module>core</module>
      <module>site</module>
      <module>ear</module>
   </modules>

   <!-- Build settings -->
   <build>
      <plugins>
         <plugin>
            <artifactId>maven-assembly-plugin</artifactId>
            <version>2.1</version>
            <configuration>
               <descriptors>
                  <descriptor>./src/build/assembly.xml</descriptor>
               </descriptors>
               <finalName>${artifactId}-dist</finalName>
               <outputDirectory>target/site/downloads</outputDirectory>
            </configuration>
            <executions>
               <execution>
```

```
                    <phase>site</phase>
                    <goals>
                      <goal>assembly</goal>
                    </goals>
                  </execution>
                </executions>
              </plugin>
              <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-antrun-plugin</artifactId>
                <executions>
                  <execution>
                    <id>copy-subprojects-sites</id>
                    <phase>site</phase>
                    <configuration>
                      <tasks>
                        <!-- Copy Core's site -->
                        <mkdir dir="./target/site/bookingbooks-core" />
                        <copy todir="./target/site/bookingbooks-core">
                          <fileset dir="./core/target/site" />
                        </copy>
                        <!-- Copy Site's site -->
                        <mkdir dir="./target/site/bookingbooks-site" />
                        <copy todir="./target/site/bookingbooks-site">
                          <fileset dir="./site/target/site" />
                        </copy>
                      </tasks>
                    </configuration>
                    <goals>
                      <goal>run</goal>
                    </goals>
                  </execution>
                </executions>
              </plugin>
              <plugin>
                <groupId>net.sourceforge.xuse</groupId>
                <artifactId>maven-xuse-m2-plugin</artifactId>
                <version>00.03.05-almira</version>
                <executions>
                  <execution>
                    <phase>site</phase>
                    <id>generate-xuse-docs</id>
                    <goals>
                      <goal>xuse</goal>
                    </goals>
                  </execution>
                </executions>
              </plugin>
              <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-release-plugin</artifactId>
                <version>2.0-beta-9</version>
                <configuration>
                  <goals>deploy site-deploy</goals>
                  <generateJiraAnnouncement>false</generateJiraAnnouncement>
                  <autoVersionSubmodules>true</autoVersionSubmodules>
                  <remoteTagging>true</remoteTagging>
                </configuration>
              </plugin>
            </plugins>
          </build>
          <!-- Environment settings -->
          <distributionManagement>
```

```
            <site>
               <id>almira</id>
               <name>Almira Labs site archive</name>

<url>dav:http://devel.almiralabs.com/archive/petstore/${project.version}</url>
            </site>
            <repository>
               <id>almira</id>
               <name>Almira Labs repository</name>
               <url>dav:http://devel.almiralabs.com/repository</url>
            </repository>
         </distributionManagement>
         <ciManagement>
            <system>continuum</system>
            <url>https://devel.almiralabs.com/continuum</url>
            <notifiers>
               <notifier>
                  <type>mail</type>
                  <sendOnError>true</sendOnError>
                  <sendOnFailure>true</sendOnFailure>
                  <sendOnSuccess>false</sendOnSuccess>
                  <sendOnWarning>true</sendOnWarning>
                  <configuration>
                     <address>developers@almiralabs.com</address>
                  </configuration>
               </notifier>
            </notifiers>
         </ciManagement>
         <repositories>
            <repository>
               <id>almira</id>
               <name>Almira Labs repository</name>
               <url>http://devel.almiralabs.com/repository</url>
               <layout>default</layout>
            </repository>
         </repositories>
         <pluginRepositories>
            <pluginRepository>
               <id>almira</id>
               <name>Almira Labs repository</name>
               <url>http://devel.almiralabs.com/repository</url>
               <layout>default</layout>
            </pluginRepository>
         </pluginRepositories>
         <developers>
            <developer>
               <id>gines</id>
               <name>Ginés Sánchez</name>
               <email>gines.sanchez@almiralabs.com</email>
               <organization>Almira Labs</organization>
               <organizationUrl>http://almiralabs.com/</organizationUrl>
               <roles>
                  <role>Developer</role>
               </roles>
               <timezone>+1</timezone>
               <properties>
                  <skype>gines_g</skype>
               </properties>
            </developer>
         </developers>
      </project>
```

2.2 Core POM.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <!--
    Inherit from the almira parent pom.
  -->
  <parent>
    <groupId>com.almiralabs</groupId>
    <artifactId>almira-reports-pom</artifactId>
    <version>1.1.3</version>
  </parent>
  <inceptionYear>2008</inceptionYear>
  <name>Booking Books Core Module</name>
  <groupId>com.almiralabs.petstore</groupId>
  <artifactId>bookingbooks-core</artifactId>
  <packaging>jar</packaging>
  <!--
    This version must NEVER BE CHANGED. The release plugin will
    change it and keep it synchonized the version of every release.
  -->
  <version>1.3.4</version>
  <dependencies>
    <dependency>
      <groupId>com.almiralabs</groupId>
      <artifactId>afc-jpa</artifactId>
      <version>1.0.0</version>
      <scope>compile</scope>
    </dependency>
    <dependency>
      <groupId>com.almiralabs</groupId>
      <artifactId>afc-properties</artifactId>
      <version>1.1.0</version>
      <scope>compile</scope>
    </dependency>
    <dependency>
      <groupId>com.almiralabs.afc</groupId>
      <artifactId>afc</artifactId>
      <version>1.2.0</version>
    </dependency>
    <dependency>
      <groupId>javax.persistence</groupId>
      <artifactId>persistence-api</artifactId>
      <version>1.0</version>
      <scope>compile</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring</artifactId>
      <version>2.5.4</version>
      <scope>compile</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-mock</artifactId>
      <version>2.0.8</version>
      <scope>compile</scope>
    </dependency>
    <dependency>
```

```
            <groupId>org.aspectj</groupId>
            <artifactId>aspectjweaver</artifactId>
            <version>1.6.0</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>commons-lang</groupId>
            <artifactId>commons-lang</artifactId>
            <version>2.4</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>commons-dbcp</groupId>
            <artifactId>commons-dbcp</artifactId>
            <version>1.2.2</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>3.8.1</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-entitymanager</artifactId>
            <version>3.3.1.ga</version>
            <scope>test</scope>
        </dependency>
        <!-- Needed by hibernate-entitymanager-3.3.1.ga -->
        <dependency>
            <groupId>concurrent</groupId>
            <artifactId>concurrent</artifactId>
            <version>1.3.4</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>hsqldb</groupId>
            <artifactId>hsqldb</artifactId>
            <version>1.8.0.7</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.easymock</groupId>
            <artifactId>easymock</artifactId>
            <version>2.2</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-annotations</artifactId>
            <version>3.3.1.GA</version>
            <scope>compile</scope>
        </dependency>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>servlet-api</artifactId>
            <version>2.3</version>
        </dependency>
    </dependencies>
    <build>
        <resources>
            <resource>
```

```
            <targetPath>.</targetPath>
            <directory>${basedir}/src/main/resources</directory>
            <filtering>true</filtering>
         </resource>
         <resource>
            <targetPath>.</targetPath>
            <directory>${basedir}/src/main/config</directory>
         </resource>
      </resources>
      <testResources>
         <testResource>
            <targetPath>.</targetPath>
            <directory>${basedir}/src/test/config</directory>
         </testResource>
      </testResources>
      <plugins>
         <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>2.0.1</version>
            <configuration>
               <source>1.5</source>
               <target>1.5</target>
            </configuration>
         </plugin>
         <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-surefire-plugin</artifactId>
            <version>2.3</version>
            <configuration>
               <forkMode>pertest</forkMode>
               <testFailureIgnore>true</testFailureIgnore>
            </configuration>
         </plugin>
      </plugins>
      <extensions>
         <!-- Needed to deploy through webdav -->
         <extension>
            <groupId>org.apache.maven.wagon</groupId>
            <artifactId>wagon-webdav</artifactId>
            <version>1.0-beta-2</version>
         </extension>
      </extensions>
   </build>
   <reporting>
      <plugins>
         <plugin>
            <artifactId>maven-checkstyle-plugin</artifactId>
            <version>2.4</version>
         </plugin>
         <plugin>
            <groupId>org.codehaus.mojo</groupId>
            <artifactId>taglist-maven-plugin</artifactId>
            <version>2.2</version>
            <configuration>
               <tags>
                  <tag>TODO</tag>
                  <tag>FIXME</tag>
                  <tag>TBD</tag>
                  <tag>@todo</tag>
                  <tag>@deprecated</tag>
               </tags>
            </configuration>
```

```
            </plugin>
            <plugin>
               <groupId>org.apache.maven.plugins</groupId>
               <artifactId>maven-changes-plugin</artifactId>
               <version>2.1</version>
               <reportSets>
                  <reportSet>
                     <reports>
                        <report>changes-report</report>
                     </reports>
                  </reportSet>
               </reportSets>
               <configuration>
                  <issueLinkTemplate>%URL%/%ISSUE%</issueLinkTemplate>
               </configuration>
            </plugin>
            <plugin>
               <groupId>org.apache.maven.plugins</groupId>
               <artifactId>maven-javadoc-plugin</artifactId>
               <version>2.5</version>
               <configuration>
                  <doclet>org.jboss.apiviz.APIviz</doclet>
                  <docletArtifact>
                     <groupId>org.jboss.apiviz</groupId>
                     <artifactId>apiviz</artifactId>
                     <version>1.3.0.GA</version>
                  </docletArtifact>
                  <useStandardDocletOptions>true</useStandardDocletOptions>
                  <charset>UTF-8</charset>
                  <encoding>UTF-8</encoding>
                  <docencoding>UTF-8</docencoding>
                  <breakiterator>true</breakiterator>
                  <version>true</version>
                  <author>true</author>
                  <keywords>true</keywords>
                  <additionalparam>
                     -sourceclasspath
                     ${project.build.outputDirectory}
     </additionalparam>
                  </configuration>
               </plugin>
            </plugins>
         </reporting>
         <distributionManagement>
            <!--
               Needed to ensure that links to subproject's sites work ok in
               the main site stored in the archive repository.
            -->
            <site>
               <id>almira</id>
               <name>Almira Labs site archive</name>
               <!-- http://jira.codehaus.org/browse/MNG-3845 -->
               <url>file://${java.io.tmpdir}/dist</url>
            </site>
         </distributionManagement>
         <!-- Needed to be able to download parent POM -->
         <repositories>
            <repository>
               <id>almira</id>
               <name>Almira Labs repository</name>
               <url>https://devel.almiralabs.com/repository</url>
            </repository>
            <!-- JBoss.org repository -->
```

```
            <repository>
               <id>jboss.releases</id>
               <name>JBoss releases</name>
               <url>http://repository.jboss.org/maven2</url>
               <releases>
                  <enabled>true</enabled>
               </releases>
               <snapshots>
                  <enabled>false</enabled>
               </snapshots>
            </repository>
         </repositories>
      </project>
```

2.2 Ear POM.xml

```
      <project                                    xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
         <modelVersion>4.0.0</modelVersion>
         <!--
           Inheritance definition.
        -->
         <parent>
            <groupId>com.almiralabs</groupId>
            <artifactId>almira-reports-pom</artifactId>
            <version>1.1.3</version>
         </parent>
         <name>Booking Books EAR Module</name>
         <groupId>com.almiralabs.petstore</groupId>
         <artifactId>bookingbooks-ear</artifactId>
         <packaging>ear</packaging>
         <!--
            This version must NEVER BE CHANGED. The release plugin will
            change it and keep it synchonized the version of every release.
         -->
         <version>1.3.4</version>
         <dependencies>
            <dependency>
               <groupId>${project.groupId}</groupId>
               <artifactId>bookingbooks-site</artifactId>
               <version>${project.version}</version>
               <scope>compile</scope>
               <type>war</type>
            </dependency>
         </dependencies>
         <build>
            <plugins>
               <plugin>
                  <groupId>org.apache.maven.plugins</groupId>
                  <artifactId>maven-ear-plugin</artifactId>
                  <version>2.2</version>
                  <configuration>
                     <modules>
                        <webModule>
                           <groupId>com.almiralabs.petstore</groupId>
                           <artifactId>bookingbooks-site</artifactId>
                           <bundleFileName>site.war</bundleFileName>
                           <contextRoot>/bookingbooks</contextRoot>
                        </webModule>
                     </modules>
```

```
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-eclipse-plugin</artifactId>
                <configuration>
                    <additionalProjectnatures>
                        <projectnature>
                            org.eclipse.wst.common.modulecore.ModuleCoreNature
            </projectnature>
                        <projectnature>
                            org.eclipse.wst.common.project.facet.core.nature
            </projectnature>
                    </additionalProjectnatures>
                    <additionalBuildcommands>
                        <buildcommand>
                            org.eclipse.wst.validation.validationbuilder
            </buildcommand>
                        <buildcommand>
                            org.eclipse.wst.common.project.facet.core.builder
            </buildcommand>
                    </additionalBuildcommands>
                </configuration>
            </plugin>
        </plugins>
    </build>
    <distributionManagement>
        <!--
            Needed to ensure that links to subproject's sites work ok in
            the main site stored in the archive repository.
        -->
        <site>
            <id>almira</id>
            <name>Almira Labs site archive</name>
            <!-- http://jira.codehaus.org/browse/MNG-3845 -->
            <url>file://${java.io.tmpdir}/dist</url>
        </site>
    </distributionManagement>
    <!-- Needed to be able to download parent POM -->
    <repositories>
        <repository>
            <id>almira</id>
            <name>Almira Labs repository</name>
            <url>https://devel.almiralabs.com/repository</url>
        </repository>
    </repositories>
</project>
```

2.3 Site POM.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>com.almiralabs</groupId>
        <artifactId>almira-reports-pom</artifactId>
        <version>1.1.4</version>
    </parent>
    <name>Booking Books Site Module</name>
    <groupId>com.almiralabs.petstore</groupId>
```

```
<artifactId>bookingbooks-site</artifactId>
<packaging>war</packaging>
<!--
   This version must NEVER BE CHANGED. The release plugin will
   change it and keep it synchonized the version of every release.
-->
<version>1.3.4</version>
<dependencies>
   <dependency>
      <groupId>${project.groupId}</groupId>
      <artifactId>bookingbooks-core</artifactId>
      <version>${project.version}</version>
      <scope>compile</scope>
   </dependency>
   <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring</artifactId>
      <version>2.5.4</version>
      <scope>compile</scope>
   </dependency>
   <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>servlet-api</artifactId>
      <version>2.3</version>
      <scope>provided</scope>
   </dependency>
   <dependency>
      <groupId>struts</groupId>
      <artifactId>struts</artifactId>
      <version>1.2.9</version>
      <scope>compile</scope>
   </dependency>
   <dependency>
      <groupId>com.lowagie</groupId>
      <artifactId>itext</artifactId>
      <version>1.3.1</version>
   </dependency>
   <dependency>
      <groupId>org.xhtmlrenderer</groupId>
      <artifactId>core-renderer</artifactId>
      <version>R8pre2</version>
   </dependency>
   <dependency>
      <groupId>strutstestcase</groupId>
      <artifactId>strutstestcase</artifactId>
      <version>2.1.4-1.2-2.4</version>
      <scope>test</scope>
   </dependency>
   <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
   </dependency>
   <dependency>
      <groupId>org.easymock</groupId>
      <artifactId>easymock</artifactId>
      <version>2.2</version>
      <scope>test</scope>
   </dependency>
</dependencies>
<!-- Build settings -->
<build>
```

```
        <resources>
          <resource>
            <targetPath>.</targetPath>
            <directory>${basedir}/src/main/resources</directory>
          </resource>
          <resource>
            <targetPath>.</targetPath>
            <directory>${basedir}/src/main/config</directory>
          </resource>
        </resources>
        <testResources>
          <testResource>
            <targetPath>.</targetPath>
            <directory>${basedir}/src/test/groovy</directory>
          </testResource>
          <testResource>
            <targetPath>.</targetPath>
            <directory>${basedir}/src/main/webapp</directory>
          </testResource>
        </testResources>
        <plugins>
          <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>2.0.1</version>
            <configuration>
              <source>1.5</source>
              <target>1.5</target>
            </configuration>
          </plugin>
          <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-eclipse-plugin</artifactId>
            <configuration>
              <wtpversion>1.0</wtpversion>
            </configuration>
          </plugin>
          <plugin>
            <groupId>net.sourceforge.xuse</groupId>
            <artifactId>maven-xuse-m2-plugin</artifactId>
            <version>00.03.05-almira</version>
            <executions>
              <execution>
                <phase>site</phase>
                <id>generate-xuse-docs</id>
                <goals>
                  <goal>xuse</goal>
                </goals>
              </execution>
            </executions>
          </plugin>
        </plugins>
      </build>
      <reporting>
        <plugins>
          <plugin>
            <artifactId>maven-checkstyle-plugin</artifactId>
            <version>2.4</version>
          </plugin>
          <plugin>
            <groupId>org.codehaus.mojo</groupId>
            <artifactId>taglist-maven-plugin</artifactId>
            <version>2.2</version>
```

```
            <configuration>
              <tags>
                <tag>TODO</tag>
                <tag>FIXME</tag>
                <tag>TBD</tag>
                <tag>@todo</tag>
                <tag>@deprecated</tag>
              </tags>
            </configuration>
          </plugin>
          <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-changes-plugin</artifactId>
            <version>2.1</version>
            <reportSets>
              <reportSet>
                <reports>
                  <report>changes-report</report>
                </reports>
              </reportSet>
            </reportSets>
            <configuration>
              <issueLinkTemplate>%URL%/%ISSUE%</issueLinkTemplate>
            </configuration>
          </plugin>
          <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-javadoc-plugin</artifactId>
            <version>2.5</version>
            <configuration>
              <doclet>org.jboss.apiviz.APIviz</doclet>
              <docletArtifact>
                <groupId>org.jboss.apiviz</groupId>
                <artifactId>apiviz</artifactId>
                <version>1.3.0.GA</version>
              </docletArtifact>
              <useStandardDocletOptions>true</useStandardDocletOptions>
              <charset>UTF-8</charset>
              <encoding>UTF-8</encoding>
              <docencoding>UTF-8</docencoding>
              <breakiterator>true</breakiterator>
              <version>true</version>
              <author>true</author>
              <keywords>true</keywords>
              <additionalparam>
                -sourceclasspath
                ${project.build.outputDirectory}
          </additionalparam>
            </configuration>
          </plugin>
        </plugins>
      </reporting>
      <distributionManagement>
        <!--
          Needed to ensure that links to subproject's sites work ok in
          the main site stored in the archive repository.
        -->
        <site>
          <id>almira</id>
          <!-- http://jira.codehaus.org/browse/MNG-3845 -->
          <url>file://${java.io.tmpdir}/dist</url>
        </site>
      </distributionManagement>
```

```
        <!-- Needed to be able to download parent POM -->
        <repositories>
          <repository>
            <id>almira</id>
            <name>Almira Labs repository</name>
            <url>https://devel.almiralabs.com/repository</url>
          </repository>
          <!-- JBoss.org repository -->
          <repository>
            <id>jboss.releases</id>
            <name>JBoss releases</name>
            <url>http://repository.jboss.org/maven2</url>
            <releases>
              <enabled>true</enabled>
            </releases>
            <snapshots>
              <enabled>false</enabled>
            </snapshots>
          </repository>
        </repositories>
      </project>
```

7.5.1.3. Replication

ISO 90003:2004 Guideline

Where required, the organization should establish and perform replication, considering the following to ensure that replication is conducted correctly:

a) identification of the master and the copies, including format, variant and version;

b) the type of media for each software item and associated labelling;

c) the stipulation of required documentation such as manuals, user guides, licenses and release notes, including identification and packaging;

d) controlling the environment under which the replication is effected to ensure repeatability;

e) provision for ensuring correctness and completeness of the copies of the product.

Guideline Application

1. Description

Replication issues are resolved with the Software Configuration Management (SCM). SCM is the management of changes to documents, programs, and other information stored as computer files.

I have used *Subversion* as SCM tool. It is necessary when a team of people may change the same files. Changes are usually identified by numbers. For example, an initial set of files is "revision 1". When the first change is made, the resulting set is "revision 2", and so on. Each revision is associated with a timestamp and the person making the change. Revisions can be compared, restored, and with some types of files, merged.

Engineering revision control developed from formalized processes based on tracking revisions of early blueprints. This system of control implicitly allowed returning to any earlier state of the design, for cases in which an engineering dead-end was reached in the development of the design. Likewise, in computer software engineering, revision control is any practice that tracks and provides control over changes to source code.

I have used revision control in *Booking Books* to maintain documentation and configuration files as well as source code.

Guideline Application Tool

1. Description

SCM issues are done with *Subversion*.

7.5.1.4. Delivery

ISO 90003:2004 Guideline

Delivery may be achieved by physical movement of media containing software or by electronic transmission.

The preservation of items during delivery is covered in 7.5.5.

Guideline Application

1. Description

Delivery is the process of packaging and delivering a software product so that it can be installed on a user's computer. The two major methods of delivering software are manual installs and managed installs.

- **Manual install** is the preferred delivery solution because it offers the simplest install experience for small or compact

products, such as a single application package. For example, to install an application, a user may drag the application package from a CD onto a folder of their choosing.

- **Managed installs** let you define every aspect of the install experience, including making sure the target computer meets specific requirements. Managed installs are generally used with products comprising several components to tailor the installation of each component depending on its kind. A managed install uses installer packages that define an install experience. When users open such packages, the Installer application guides them through the installation process and copies the product files to the appropriate locations on their file system.

Guideline Application Tool

1. Description

It is not necessary add any guideline application tool.

7.5.1.5. Installation

ISO 90003:2004 Guideline

Sometimes, customers or third parties conduct installation. In this case the role of the organization is to describe the steps the customer, or third party, needs to take to perform the installation. Sometimes, the installation is conducted by the organization. For the latter case, the following may apply:

a) the organization and customer should agree on their respective roles, responsibilities and obligations;

b) the need and extent of validation at each installation should be defined;

c) the need for installation instructions should be defined;

d) the need for configuration of the software and hardware for the specific installation should be defined;

e) the need for data capture and/or conversion and database population should be defined;

f) the acceptance procedure of each installation upon completion should be defined;

g) a schedule is needed;

h) access to customer's facilities and equipment should be arranged (e.g. security badges, passwords, escorts);

i) the availability of skilled personnel should be established;

j) the need to provide training associated with the specific intended use of the product during installation or as part of maintenance should be defined;

k) the need to perform backup and confirm recovery should be defined.

The introduction of a new software product or new software release at multiple user sites can require planning of implementation or rollout.

Guideline Application

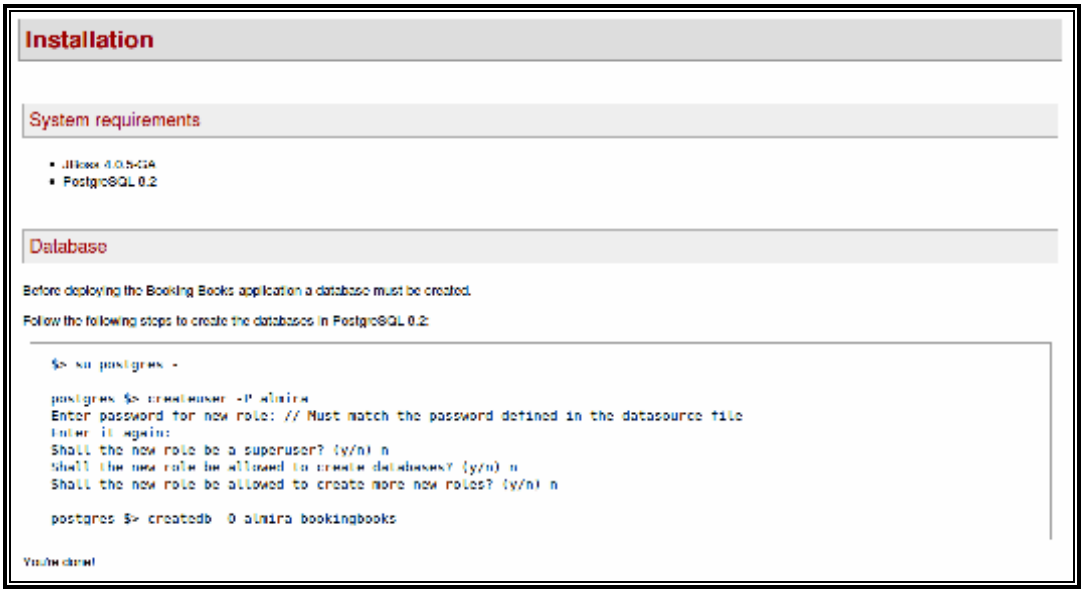
1. Description

In Booking Books project, installation guide is described as *Xuse* xml files.

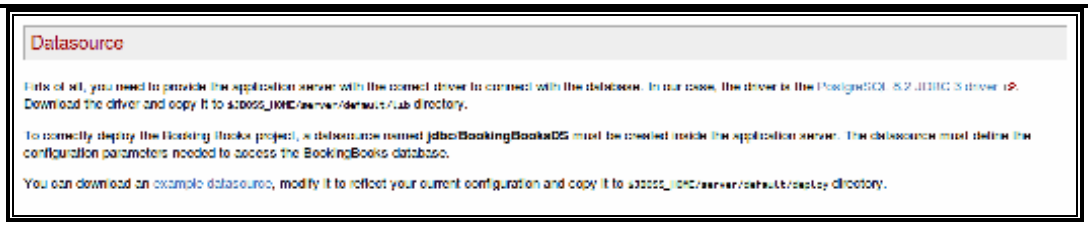
Guideline Application Tool

1. Description

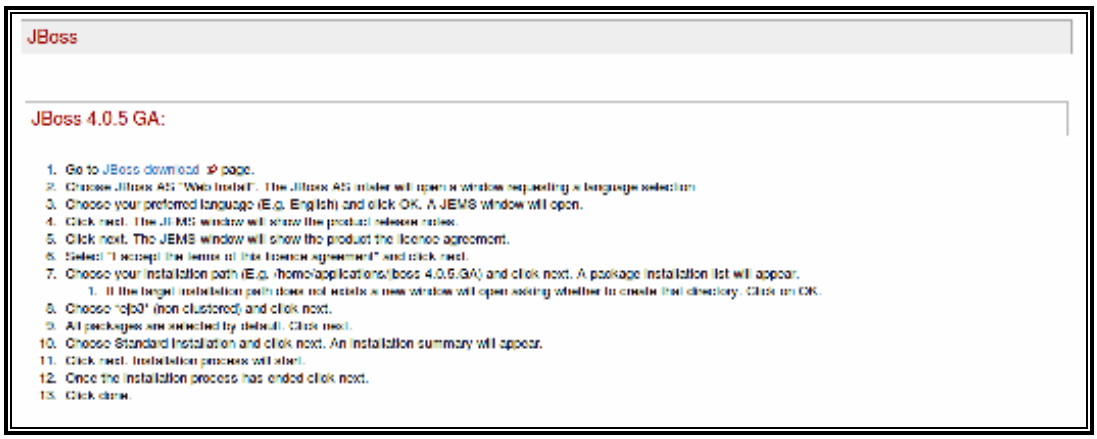
It is not necessary add any guideline application tool.



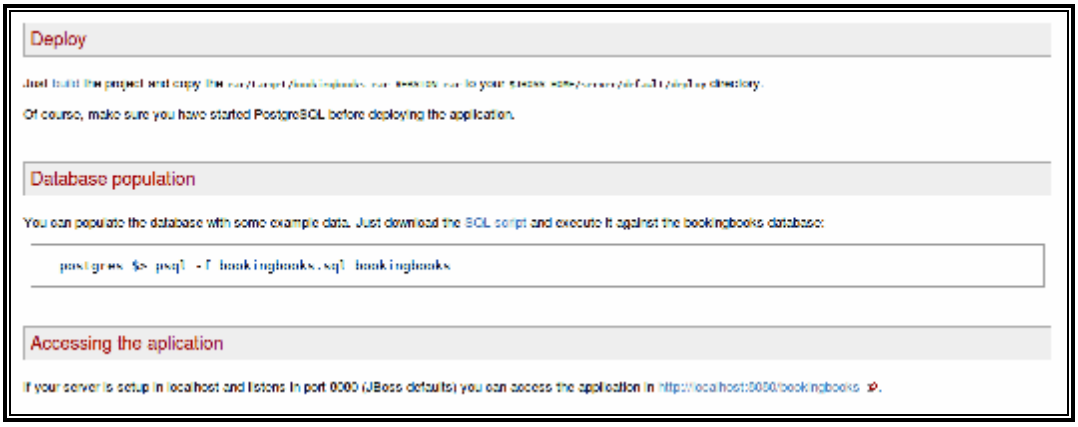
Picture III-127: Installation Template



Picture III-128: Datasource Configuration



Picture III-129: JBoss Installation Setup



Picture III-130: Deploy and Database Population

7.5.1.6. Operations

ISO 90003:2004 Guideline

A software-producing organization should plan and control operations, including

a) the need to set up a help desk to conduct telephone or other electronic communication with the customer(s), and

b) arrangements for ensuring continuity of support, such as disaster recovery, security and backup (see 6.3).

Guideline Application

1. Description

It is not necessary add any guideline. Arrangements for ensuring continuity of support are beyond the scope of this project.

Guideline Application Tool

1. Description

It is not necessary add any guideline application tool.

7.5.1.7. Maintenance

ISO 90003:2004 Guideline

Maintenance of the software product that is requested by the customer for specific items, and a specific period of time, after initial delivery and installation, should be stipulated in the contract. The organization should establish a process for performing maintenance activities and verifying them. Maintenance activities may also be performed on the development environment, tools and documentation. Maintenance should include the following, as appropriate:

a) scope of maintenance;

b) identification of the initial status of the maintained items;

c) support organization(s) and arrangements (see also 7.5.1.6);

d) maintenance activities including problem resolution help desk support, hardware support and system monitoring to detect failure;

e) interface modifications that may be required when additions or changes are made to the hardware system, or components, controlled by the software;

f) configuration management, testing and quality assurance activities;

g) proposed release schedule;

h) how functional expansion and performance improvement will be carried out;

i) maintenance records and reports.

The records of the maintenance activities may be utilized for evaluation and enhancement of the software product and for improvement of the quality management system itself. When resolving problems, temporary fixes may be used to minimize downtime and permanent modifications carried out later.

For interface modifications and functional expansion, depending upon the scale of work, change control procedures should apply, or a new and separate development project should be initiated.

NOTE For further information, see ISO/IEC 12207:1995[11], 5.3.12 (software installation), 5.4.4 (user support), 5.5 (maintenance process), 6.6.3 (process assurance) and 6.8 (problem resolution process), and ISO/IEC 12207:1995/Amd.1:2002[12], F.1.3.11 (software installation), F.1.4.2 (customer support), F.1.5 (maintenance process) and F.2.8 (problem resolution process).

Guideline Application

1. Description

It is not necessary add any guideline. Maintenance issues are beyond the scope of this project.

Guideline Application Tool

1. Description

It is not necessary add any guideline application tool.

7.5.2. Validation of process for production and service provision

ISO 9001:2008 Standard

The organization shall validate any processes for production and service provision where the resulting output cannot be verified by subsequent monitoring or measurement and, as a consequence, deficiencies become apparent only after the product is in use or the service has been delivered.

Validation shall demonstrate the ability of these processes to achieve planned results.

The organization shall establish arrangements for these processes including, as applicable

a) defined criteria for review and approval of the processes,

b) approval of equipment and qualification of personnel,

c) use of specific methods and procedures,

d) requirements for records (see 4.2.4), and

e) revalidation.

ISO 90003:2004 Guideline

The organization should consider what processes may be used to compensate for the inability to validate fully the product. Examples include the following:

a) a design and development review might consider how the design and development might fail in addition to the more normal check that the design and development will function correctly;

b) a program of failure mode and effect analyses that builds up a history of design and development failures and how they can be avoided.

Whatever methods are used, they should be commensurate with the risks and consequences of design and development failures.

Guideline Application

1. Description

It is not necessary add any guideline.

Guideline Application Tool

1. Description

It is not necessary add any guideline application tool.

7.5.3. Identification and traceability

ISO 9001:2008 Standard

Where appropriate, the organization shall identify the product by suitable means throughout product realization.

The organization shall identify the product status with respect to monitoring and measurement requirements through product realization.

Where traceability is a requirement, the organization shall control the unique identification of the product and maintain records (see 4.2.4).

NOTE In some industry sectors, configuration management is a means by which identification and traceability are maintained.

7.5.3.1. Overview

ISO 90003:2004 Guideline

For software, identification and traceability is commonly implemented through configuration management.

Configuration management is a management discipline that applies technical and administrative direction to the design, development and support of configuration items, including software items. This discipline is also applicable to related documentation (see also 4.2.3) and hardware. The degree of configuration management use is dependent on the project size, complexity and risk level.

One objective of configuration management is to provide full visibility of the product's present configuration and status. Another objective is that everyone working on the product at any time in its life cycle uses appropriate versions of items.

Guideline Application

1. Description

Traceability is done by configuration management (with *Subversion*). See *7.5.1.2 Build and Release*.

Guideline Application Tool

1. Description

See *7.5.1.2 Build and Release*.

7.5.3.2. Configuration management process

ISO 90003:2004 Guideline

The scope of configuration management should include the following:

a) planning of the process including defining activities, responsibilities and the tools to be procured;

b) identifying uniquely the name and versions of each configuration item and when they are to be brought under configuration control (configuration identification);

c) identifying the versions of each software item which together constitute a specific version of a complete product (baseline), including re-used software, libraries, and purchased and customer supplied software;

d) identifying the build status of software products under development, delivered or installed, for single or multiple environments, as appropriate;

e) controlling simultaneous updates of a given software item by two or more people working independently (configuration control);

f) providing coordination for the updating of multiple products in one or more locations as required;

g) identifying, tracking and reporting of the status of items, including all actions and changes resulting from a change request or problem, from initiation through to release (configuration status accounting);

h) providing configuration evaluation (status of verification and validation activities);

i) providing release management and delivery.

Guideline Application

1. Description

Configuration management issues are done with subversion, and maven is used as a project management and comprehension tool, providing release management and delivery.

In the POM.xml files of maven are defined the name and version of each configuration item, libraries and frameworks used to develop the software, the software version or version which will be released, and so on. To see an example of the maven configuration file, see *7.5.1.2. Build and release*.

Guideline Application Tool

1. Description

See *7.5.1.2. Build and release*.

7.5.3.3. Traceability

ISO 90003:2004 Guideline

Throughout the product life cycle, there should be a process to trace the components of the software item or product. Such tracing may vary in scope according to the requirements of the contract or marketplace, from being able to place a certain change request in a specific release, to recording the destination and usage of each variant of the product.

NOTE For further information, see the following:

— ISO 10007[9] (guidelines for configuration management);

— ISO/IEC 12207:1995[11], 6.2, and ISO/IEC 12207:1995/Amd.1:2002[12], F.2.2 (configuration management process);

— ISO/IEC TR 15846:1998[27] (software life cycle processes — configuration management), Clauses 7 to 12.

Guideline Application

1. Description

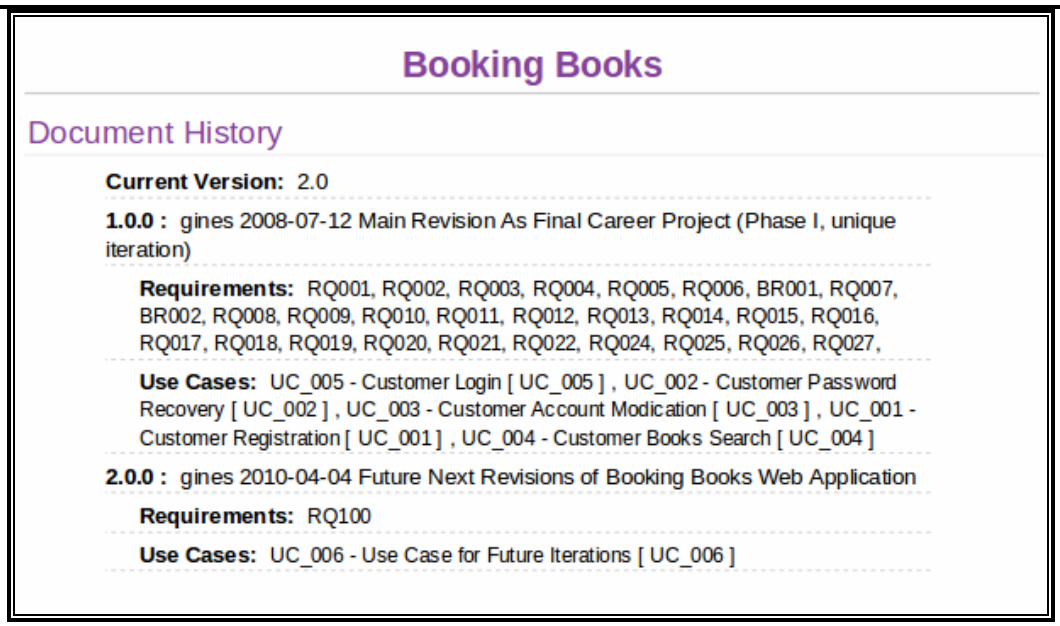
Traceability is the ability to verify the history, location, or application of an item by means of documented recorded identification.

I have done traceability from requirements to use cases for every software version.

Guideline Application Tool

1. Description

In *Xuse* I have recorded the traceability between software versions, use cases and requirements.



Picture III-131: Document History

7.5.4. Customer property

ISO 9001:2008 Standard

The organization shall exercise care with customer property while it is under the organization's control or being used by the organization. The organization shall identify, verify, protect and safeguard customer property provided for use or incorporation into the product. If any customer property is lost, damaged or otherwise found to be unsuitable for use, this organization shall report this to the customer and maintain records.

ISO 90003:2004 Guideline

The organization may be required to acquire and include product and data supplied by the customer, e.g.

a) software products including commercial software products supplied by the customer,

b) development tools,

c) development environments including network services,

d) test and operational data,

e) interface or other specifications,

f) hardware, and

g) intellectual property, and confidential and proprietary information, including specifications.

In any maintenance agreement consideration should be given to addressing

— required licensing and support, including subsequent revisions to the product, and

— limitations or constraints in re-use of the product in other projects.

The means by which updates to customer-supplied items are accepted and integrated should be defined. The organization may apply the same kinds of verification activities to customer-supplied product as to purchased products. This includes requirements for records indicating which changes have been implemented, and at what locations for multiple products and sites.

The methods for identifying the customer-supplied product should be part of configuration management for the product.

Guideline Application

1. Description

Customer property include any materials, parts, components, etc. that are provided by the customer to be incorporated into the organization's product. We can also include any property that is used by the organization, such as customer-provided equipment, tools, hardware and software. Such product may be owned directly by the customer, or owned by another interested party (as software libraries or dependencies).

Intellectual property may also fall under this requirement, if its return is required upon completion of a project. If it is given freely however (e.g., public information) it would not fall under this requirement.

Guideline Application Tool

1. Description

It is not necessary add any guideline application tool.

7.5.5. Preservation of product

ISO 9001:2008 Text

The organization shall preserve the product during internal processing and delivery to the intended destination in order to maintain conformity to requirements. As applicable, preservation shall include identification, handling, packaging, storage and protection. Preservation shall also apply to the constituent parts of a product.

ISO 90003:2004 Guideline

A software-producing organization should ensure that its products are not altered from the point of production, through replication, handling and storage, to the point of delivery. Software information does not degrade; however, the media on which it is stored may be subject to deterioration, and appropriate precautions should be taken by the organization.

Delivery should provide for appropriate preventive action to protect the software product from damage. In addition, an appropriate level of software virus checking and appropriate measures to protect product integrity are needed. Delivery of software may be achieved by physical movement of media containing software, or by electronic transmission. The following should be considered and appropriate actions taken when handling, packaging, storing or delivering software:

a) storing software items, maintaining versions of products in established baselines;

b) permitting the controlled access to and retrieval of the master and any copies, protecting them from unauthorized change or corruption;

c) protecting computer media, particularly with respect to computer viruses, electromagnetic and electrostatic environments;

d) providing for regular backup of software, including off-site storage for disaster recovery;

e) ensuring the timely copying of software to replacement media;

f) storing of software media in a protected environment, preventing deterioration and protecting from obsolescence;

g) the effects of using compression and decompression techniques (the reduction of the space taken on a data medium by encoding data, taking advantage of redundancy in the data);

h) the effects of using encryption and decryption techniques (the transformation of data into an unintelligible form for data security).

NOTE For further general guidance related to ISO 9001:2000, 7.5, see the following:

— ISO/IEC 9126-1:2001[5] for guidance on quality characteristics of software products;

— ISO/IEC TR 15846:1998[27];

— ISO/IEC 14764:1999[19];

— ISO/IEC 15910:1999[28].

Guideline Application

1. Description

Preservation of product can be done with a cryptography hash-function like MD5 (Message-Digest algorithm 5). MD5 has been employed in a wide variety of security applications, and is also commonly used to check the integrity of files. An MD5 hash is typically expressed as a 32-digit hexadecimal number.

Guideline Application Tool

1. Description

There are many tools to verify the MD5 hash (like for example the program *md5sum* which is installed by default in *Ubuntu*).

The program md5sum is designed to verify data integrity using the MD5 128-bit cryptographic hash. MD5 hashes used properly can confirm both file integrity and authenticity.

In terms of integrity, an MD5 hash comparison detects changes in files that would cause errors. The possibility of changes (errors) is proportional to the size of the file; the possibility of errors increase as the file becomes larger. It is a very good idea to run an MD5 hash comparison check when you have a file like *Booking Books*.

Booking Books will give to the client the MD5 hash signed from a secure source (an HTTPS page). While security flaws in the MD5 algorithm have been uncovered, MD5 hashes are still useful when you trust the organization that produces them.

7.6. Control of monitoring and measuring devices

ISO 9001:2008 Text

The organization shall determine the monitoring and measurement to be undertaken and the monitoring and measuring equipment needed to provide evidence of conformity of product to determined requirements.

The organization shall establish processes to ensure that monitoring and measurement can be carried out and are carried out in a manner that is consistent with the monitoring and measurement requirements.

Where necessary to ensure valid results, measuring equipment shall

a) be calibrated or verified at specified intervals, or prior to use, against measurement standards traceable to international or national measurement standards; where no such standards exist, the basis used for calibration or verification shall be recorded (see 4.2.4);

b) be adjusted or re-adjusted as necessary;

c) have identification in order to determine its calibration status;

d) be safeguarded from adjustments that would invalidate the measurement result;

e) be protected from damage and deterioration during handling, maintenance and storage.

In addition, the organization shall assess and record the validity of the previous measuring results when the equipment is found not to conform to requirements. The organization shall take appropriate action on the equipment and any product affected.

Records of the results of calibration and verification shall be maintained (see 4.2.4).

When used in the monitoring and measurement of specified requirements, the ability of computer software to satisfy the intended application shall be confirmed. This shall be undertaken prior to initial use and reconfirmed as necessary.

NOTE Confirmation of the ability of computer software to satisfy the intended application would typically include its verification and configuration management to maintain its suitability for use.

ISO 90003:2004 Guideline

Calibration is a technique that often has been perceived as not directly applicable to software. However, it may be applicable to hardware and tools used to test and validate the software. Consequently, items a) to e) in ISO 9001:2008, 7.6, may be applicable to the environment used when testing the software.

Where the organization uses tools, facilities and techniques in the conduct of any tests verifying conformance of the software product to specified requirements, the organization should consider the effect of such tools on the quality of the software product, when approving them. In addition, such tools may be placed under configuration management prior to use.

Although *adjusted or re-adjusted as necessary* [ISO 9001:2008, 7.6, item b)] is not applicable to software, there may be a need to verify periodically that software used in measuring devices has not changed, due to exposure to harsh environments, such as viruses or electromagnetic fields.

The suitability of test tools, techniques and data should be verified prior to use, to determine if there is a need to improve and/or upgrade them. The organization should have procedures for determining how the test software is checked.

Measuring and monitoring devices used in software development, testing, maintenance and operation include

a) data used for testing the software product,

b) software tools (e.g. for simulation, collecting performance, resource utilization and coverage information),

c) computer hardware, and

d) instrumentation interfacing to the computer hardware.

The organization should control measuring and monitoring devices by means of a configuration management system (see 7.5.3).

Guideline Application

1. Description

It is not necessary add any guideline.

Guideline Application Tool

1. Description

It is not necessary add any guideline application tool.

8. Measurement, Analysis and Improvement

8.1. General

ISO 9001:2008 Standard

The organization shall plan and implement the monitoring, measurement, analysis and improvement processes needed

a) to demonstrate conformity to product requirements,

b) to ensure conformity of the quality management system, and

c) to continually improve the effectiveness of the quality management system.

This shall include determination of applicable methods, including statistical techniques, and the extent of their use.

ISO 90003:2004 Guideline

The purpose of the software measurement process is to collect, analyse and report data relating to the products developed and processes implemented within the organizational unit, to support effective management of the processes, and to demonstrate objectively the quality of the products.

The monitoring, measurement, analysis and improvement processes should be identified as part of quality planning (see 7.1.2).

NOTE For further information, see the following:

— ISO/IEC 12207:1995[11], 7.3, and ISO/IEC 12207:1995/Amd.1:2002[12], F.3.3 (improvement process);

— ISO/IEC 15939:2002[29], Clause 5 (software measurement process);

— ISO/IEC 15504-1[22];

— ISO/IEC TR 9126-2[6] and ISO/IEC TR 9126-3[7] (product quality — internal and external metrics);

— ISO/IEC 14598-2[14] (software product evaluation — planning and management).

Guideline Application

1. Description

Measurement, analysis and improvement processes are vital to the achievement of quality. Until we measure using devices of known integrity, we know little about a process or its outcomes.

But if we measure using instruments that are unfit for purpose, we will be misled by the results. With the results of valid measurement we can make a judgement on the basis of facts. The facts will tell us whether we have met the target. Analysis of the facts will tell us whether the target can be met using the same methods or better methods or whether the target is the right target to aim for.

Measurements without a target value to compare results of measurement are measurements without a purpose. The target value is therefore vital but arbitrary values unmotivate personnel. Targets should always be focused on purpose so that through the chain of measures from corporate objectives to component dimensions there is a soundly based relationship between targets, measures, objectives and the purpose of the organization, process or product.

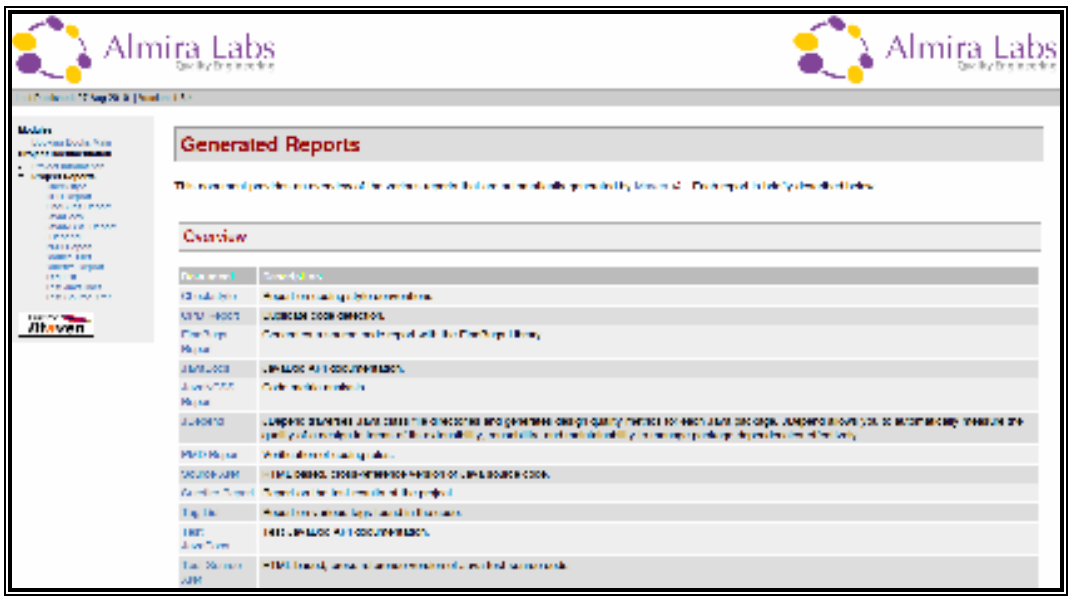
Measurement tells us whether there has been a change in performance. Change is a constant. It exists in everything and is caused by physical, social or economic forces. When we measure the same parameter on different items we expect slight variation. However, if we measure the same parameter using the same device we might not expect there to be a change, but the inaccuracies inherent in the measuring system will lead to a variation in readings. To understand change we need to understand its cause. Some change is represented by variation about a norm and is predictable – it is a natural phenomenon of a process and when it is within acceptable limits it is tolerable. Other change is represented by erratic behaviour and is not predictable but its cause can be determined and eliminated through measurement, analysis and improvement.

Measurement, analysis and improvement are strictly sub-processes within each business process. However, parent processes will often capture data from monitoring and measurements within sub-processes. This may happen when assessing a variety of data from individual processes to determine customer satisfaction or for discovering common cause problems and subsequently devising company-wide improvement programmes.

Measurement, analysis and improvement can be done using metric reports.

1. Description

For metric reports I will use *Maven*.



Picture III-132: Generated Metric Reports with Maven

8. 2 Monitoring and measurement

8.2.1 Customer satisfaction

ISO 9001:2008 Standard

As one of the measurements of the performance of the quality management system, the organization shall monitor information relating to customer perception as to whether the organization has met customer requirements. The methods for obtaining and using this information shall be determined.

NOTE Monitoring customer perception can include obtaining input from sources such as customer satisfaction surveys, customer data on delivered product quality, user opinion surveys, lost business analysis, compliments, warranty claims and dealer reports.

ISO 90003:2004 Guideline

The organization's process for requesting, measuring and monitoring feedback of customer satisfaction should provide information on a

continual or periodic basis as appropriate. For software consider, for example,

a) analysis of help desk calls relating to both product quality and service performance,

b) quality-in-use metrics derived from customer direct and indirect feedback,

c) other quality metrics based on use of the product, and

d) number of software releases needed to fix problems, after initial delivery.

NOTE For further information, see ISO/IEC TR 9126-4[8] (product quality — quality-in-use metrics).

Guideline Application

1. Description

Customer satisfaction is an essential element to staying in business in this modern world of global competition. We must satisfy and even delight our customers with the value of our software products and services to gain their loyalty and repeat business. Customer satisfaction is therefore a primary goal of process improvement programs.

So how satisfied are our customers? One of the best ways to find out is to ask them using Customer Satisfaction Surveys. These surveys can provide management with the information they need to determine their customer's level of satisfaction with their software products and with the services associated with those products. Software engineers and other members of the technical staff can use the survey information to identify opportunities for ongoing process improvements and to monitor the impact of those improvements.

This guideline includes details on designing your own software customer satisfaction questionnaire, tracking survey results and example reports that turn survey data into useful information.

2. Focusing on Key Customer Quality Requirements

When creating a Customer Satisfaction Survey, our first objective is to get the customer to participate. If the survey deals with issues that the customer cares about, they are more likely to participate. We also want to make sure that the survey is short and easy to complete in order to increase our chances of this happening. If the survey is long and detailed, the recipient is more likely to set it aside to complete later, only to have it disappear into

the stacks of other papers on their desk. Therefore, the first step in creating a Customer Satisfaction Survey is to focus in on the customer's key quality requirements.

When determining this list of key quality requirements it can be helpful to select those factors that are relevant to your specific products or services. An example is the *ISO 9126* standard, *Information Technology - Software Product Evaluation - Quality Characteristics and Guidelines for Their Use*, that defines seven quality characteristics for software product evaluation including:

- Usability
- Reliability
- Efficiency
- Reusability
- Maintainability
- Portability
- Testability

These general lists can be tailored to match the quality requirements of a specific software product or service.

For example, if the software product has extensive user interfaces and is sold internationally, the ability to easily change the product to meet the needs of languages other than English may be a key quality requirement. An excellent source of information to use when making a tailoring decision is the people who created the software product or who provide the software services. They can have unique insight into their job functions and how they relate to meeting the customer's requirements.

Another mechanism for determining the customer's quality requirements is interview customers, and each interviewee is asked to describe 5-10 positive and 5-10 negative encounters with the product or service that they have personally encountered in the past. The incidents are then used to generate categories of *satisfaction items* based on shared common words used in the incident description. For example, both positive and negative statements about how long they had to wait for help when they phoned the technical service support line would be grouped together into a *length of wait for service* category. These satisfaction items are then used to discover key customer quality requirements. For example, the *length of wait for service* item could be combined with the *ability to schedule a convenient field representative service call appointment* item and the *number of people transferred to* item, then summarized as the quality requirement called Availability of Service.

3. Creating the Questionnaire

After selecting the key quality requirements that will be the focus of the survey, the next step is to create the survey questionnaire. The questionnaire should start with an introduction that briefly states the purpose of the survey and includes the instructions for completing the survey.

The introduction is followed by the list of questions. This survey has two questions for each of the quality requirements of functionality, usability, initial reliability, long-term reliability, technical support, installation, documentation and training. This adds redundancy to the questionnaire but it also adds a level of reliability to the survey results. Just like we would not try to determine a person's actual math aptitude by asking them a single math question, asking a single question about each quality requirement reduces the reliability of predicting the actual satisfaction level from the measured level.

The questionnaire also has two questions that judge the customers overall satisfaction, one for the software product and one for the support services.

We could have simply asked the question *Are you satisfied or dissatisfied*. However, from a statistical perspective, scales with only two response options are less reliable than scales with five response options. Studies have shown that reliability seems to level off after five scale points so further refinement of the scale does not add much value. Note that this example also asks the customer to rank the importance of each item.

See *4.2.1.28. Software Customer Satisfaction Survey* for more details.

Guideline Application Tool

1. Description

Software Customer Satisfaction Survey must be published in *Twiki*.

8.2.2 Internal audit

ISO 9001:2008 Standard

The organization shall conduct internal audits at planned intervals to determine whether the quality management system

a) conforms to the planned arrangements (see 7.1), to the requirements of this International Standard and to the quality management system requirements established by the organization, and

b) is effectively implemented and maintained.

An audit programme shall be planned, taking into consideration the status and importance of the processes and areas to be audited, as well as the results of previous audits. The audit criteria, scope, frequency and methods shall be defined. The selection of auditors and conduct of audits shall ensure objectivity and impartiality of the audit process. Auditors shall not audit their own work.

A documented procedure shall be established to define the responsibilities and requirements for planning and conducting audits, establishing records and reporting results.

Records of the audit and their results shall be maintained (see 4.2.4).

The management responsible for the area being audited shall ensure that any necessary corrections and corrective actions are taken without undue delay to eliminate detected nonconformities and their causes. Follow-up activities shall include the verification of the actions taken and the reporting of verification results (see 8.5.2).

NOTE See ISO 19011 for guidance.

ISO 90003:2004 Guideline

When software organizations separate their work into projects, audit planning should define a selection of projects and assess both the compliance of their project quality planning to the organization's quality management system and the compliance of the project to the project quality planning. This selection should ensure coverage of all stages and all processes.

This may necessitate auditing various projects at different stages of their product development life cycle, or auditing a single project as it progresses through various stages. Where the intended project changes its timescale, the internal audit schedule may be reviewed, either to change the timing of the audit or to consider a different project.

NOTE For further information, see ISO/IEC 12207:1995[11], 6.3 (quality assurance process) and 6.7 (audit process), and ISO/IEC 12207:1995/Amd.1:2002[12], F.2.3 (quality assurance process) and F.2.7 (audit process).

Guideline Application

1. Description

Internal audits will be done using a systematic methodology for analyzing business processes, procedures and activities with the goal of highlighting organizational problems and recommending solutions.

Auditor must use the planning process to evaluate business risks, the current state of controls and to determine what audit procedures you will use to test each control.

Follow the next steps:

- Review the last internal and external audit reports to understand what issues were raised, the corrective actions management committed to take, and the status of those actions. If internal audit has not previously tested whether the actions were implemented, it will need to do so during the audit project.
- Review a copy of the business's organization chart and its income statement to identify all areas of responsibility. Discuss any discrepancies between the two with business management.
- Discuss changes in the business since the last audit with management and planned changes for the near future. Management may have added new products or processes, changed key personnel, be subject to new laws or regulations, modified systems, or sold or discontinued parts of the business.

Guideline Application Tool

1. Description

Internal audit application tools are beyond the scope of this project.

8.2.3 Monitoring and measurement of process

ISO 9001:2008 Standard

The organization shall apply suitable methods for monitoring and, where applicable, measurement of the quality management system processes. These methods shall demonstrate the ability of the processes to achieve planned results. When planned results are not achieved, correction and corrective action shall be taken, as appropriate, to ensure conformity of the product.

NOTE When determining suitable methods, it is advisable that the organization consider the type and extent of monitoring or measurement appropriate to each of its processes in relation to their impact on the conformity to product requirements and on the effectiveness of the quality management system.

ISO 90003:2004 Guideline

Organizations normally measure some aspects of their processes in order to monitor, manage and assess them. The most frequent measures include

a) the planned and actual duration of a process activity,

b) the planned and actual cost of a process activity, and

c) the planned quality levels and progressive measures of the selected quality characteristics.

NOTE 1 For further information, see ISO/IEC 12207:1995[11], 7.3.2 (process assessment) and 7.3.3 (process improvement), and ISO/IEC 12207:1995/Amd.1:2002[12], F.3.3.2 (process assessment).

NOTE 2 For guidance on conducting software process assessment, see ISO/IEC 15504-1[22], and that on performing an assessment, see ISO/IEC 15504-2[23].

See also ISO/IEC 15939:2002[29], Clause 5 (software measurement process).

Guideline Application

1. Description

During the software development, is necessary plan some meetings to monitor, manage and assess all the process of the software development life cycle.

Guideline Application Tool

1. Description

These issues must be created in *Jira*, and a track must be done over them.

8.2.4 Monitoring and measurement of product

ISO 9001:2008 Standard

The organization shall monitor and measure the characteristics of the product to verify that product requirements have been met. This shall be carried out at appropriate stages of the product realization process in accordance with the planned arrangements (see 7.1). Evidence of conformity with the acceptance criteria shall be maintained.

Records shall indicate the person(s) authorizing release of product (see 4.2.4).

Product release and service delivery shall not proceed until the planned arrangements (see 7.1) have been satisfactorily completed, unless otherwise approved by a relevant authority and, where applicable, by the customer.

ISO 90003:2004 Guideline

An organization should monitor and measure the conformity of products to quality requirements by means such as review, verification and validation. Examples of product characteristics that may be monitored or measured include

a) functionality,

b) maintainability,

c) efficiency,

d) portability,

e) usability, and

f) reliability.

NOTE For further information, see the following:

— ISO/IEC 12207:1995[11], 5.3, and ISO/IEC 12207:1995/Amd.1:2002[12], F.1.3 (development process), which contains provisions for evaluation of software products during development and when completed;

— ISO/IEC 9126-1:2001[5];

— ISO/IEC 14598-3[15] and ISO/IEC 14598-5[17] (software product evaluation — process for developers and evaluators).

Guideline Application

1. Description

As the monitoring and measurement over the process, must be created issues in *Jira* to monitor and measure the conformity related to the products.

Guideline Application Tool

1. Description

These issues must be created in *Jira*, and a track must be done over them.

8.3 Control of nonconforming product

ISO 9001:2008 Standard

The organization shall ensure that product which does not conform to product requirements is identified and controlled to prevent its unintended use or delivery. A documented procedure shall be established to define the controls and related responsibilities and authorities for dealing with nonconforming product.

Where applicable, the organization shall deal with nonconforming product by one or more of the following ways:

a) by taking action to eliminate the detected nonconformity;

b) by authorizing its use, release or acceptance under concession by a relevant authority and, where applicable, by the customer;

c) by taking action to preclude its original intended use or application.

d) by taking action appropriate to the effects, or potential effects, of the nonconformity when nonconformity product is detected after delivery or use has started.

When nonconforming product is corrected it shall be subject to re-verification to demonstrate conformity to the requirements.

Records of the nature of nonconformities and any subsequent actions taken, including concessions obtained, shall be maintained (see 4.2.4).

ISO 90003:2004 Guideline

In software development, segregation of nonconforming items may be effected by transferring the item out of a production or testing

environment, into a separate environment. In the case of embedded software it may become necessary to segregate the nonconforming item (hardware) which contains the nonconforming software.

The supplier should identify at what points control and recording of nonconforming product is required. Where a software item manifests a defect during development or maintenance, the investigation and resolution of such defects should be controlled and recorded.

Configuration management may be invoked to implement part of or the whole of this requirement.

Attention should be paid to the following aspects in the disposition of nonconformities:

a) any discovered problems and their possible impacts on any other parts of the software should be noted and those responsible notified so the problems can be tracked until they are resolved;

b) areas impacted by any modifications should be identified and re-tested, and the method for determining the scope of re-testing should be identified in a documented procedure;

c) the priority of the nonconformities should be established.

With software, repair or rework to achieve fulfilment of specified requirements creates a new software version.

In software development, disposition of nonconforming product may be achieved by

a) repair or rework (i.e. to fix defects) to meet the requirement,

b) acceptance with or without repair by concession,

c) treatment as a conforming product after the amendment of requirements, and

d) rejection.

NOTE For further information, see the following:

— ISO/IEC 12207:1995[11], 6.2 (configuration management process) and 6.8 (problem resolution process), and ISO/IEC 12207:1995/Amd.1:2002[12], F.2.2 (configuration management process) and F.2.8 (problem resolution process);

— ISO/IEC 12119:1994[10];

— ISO/IEC TR 15846:1998[27].

Guideline Application

1. Description

When one or more characteristics of a product fail to meet specified requirements, it is referred to as a nonconforming product.

When a product deviates from specified product requirements, it fails to conform. Nonconformity products must be identified and controlled to prevent unintended use or delivery.

Activity	Description
Identification of non conformities.	- Identification of non conformities shall be done during conformities inspection, supervision, examination of records etc., related to works, by different levels of officers including staff for construction. - Based on the nature of non conformity observed the nature of rectification / repairs required shall be decided and appropriate instructions shall be recorded in internal audit report / inspection report (as a Jira issue). - Concerned functionaries shall ensure that required rectification / repairs are carried out.
Reporting the non-conformities by field staff.	- Wherever the conformity is of serious conformity by nature shall report the same - field staff to the top management.
Disposal.	- The Division head shall study and analyze the non conformity with reference to requirements and provide solutions, keeping in view safety, quality and financial implications. a) If the Division head feels that the non – conformity, doesn't have any substantial effect on quality of work and if it is well within the tolerance limits, and structural stability is not compromised, he may accept the deviation with correction, as feasible. - The Division head shall report all such cases to the top management for information /approval, as required. b) If Division head feels that the nature and deviation of the non – conformity can not be permitted, he shall report the same to top management. c) The top management shall study the non – conformity and suggest the necessary solution ensuring safety of structure and fulfillment of functional requirements, or decide on future course of action, such as doing the work afresh.
Record of deviations.	- Record of all deviations shall be maintained to decide upon long – term preventive actions.

Guideline Application Tool

1. Description

Issues related to non conformity products must be done using *Jira*.

8.4. Analysis of data

ISO 9001:2008 Standard

The organization shall determine, collect and analyse appropriate data to demonstrate the suitability and effectiveness of the quality management system and to evaluate where continual improvement of the effectiveness of the quality management system can be made. This shall include data generated as a result of monitoring and measurement and from other relevant sources.

The analysis of data shall provide information relating to

a) customer satisfaction (see 8.2.1),

b) conformity to product requirements (see 8.2.4),

c) characteristics and trends of processes and products, including opportunities for preventive action (see 8.2.3 and 8.2.4), and

d) suppliers (see 7.4).

ISO 90003:2004 Guideline

Examples of *analysis of data* for software may include problem reports from various levels of testing and issues identified in reviews or walkthroughs.

NOTE For further information, see the following:

— ISO/IEC 15939:2002[29], 5.4 (software measurement process — evaluate results);

— ISO/IEC 19761[31], ISO/IEC 20926[32] and ISO/IEC 20968[33].

Guideline Application

1. Description

It is not necessary add any guideline.

Guideline Application Tool

1. Description

It is not necessary add any guideline application tool.

8. 5. Improvement

8.5.1. Continual improvement

ISO 9001:2008 Standard

The organization shall continually improve the effectiveness of the quality management system through the use of the quality policy, quality objectives, audit results, analysis of data, corrective and preventive actions and management review.

ISO 90003:2004 Guideline

A strategic approach to process improvement may be achieved by establishing an improvement process. This can be applied to any or all of the software life cycle processes and involves process establishment, process assessment and process improvement.

NOTE For further information, see the following:

— ISO/IEC 12207:1995[11], 7.3, and ISO/IEC 12207:1995/Amd.1:2002[12], F.3.3 (improvement process);

— ISO/IEC 15504 (all parts) (software process assessment).

Guideline Application

1. Description

The best way to perform a continual improvement is performing a maturity model.

For software developments, the most important maturity models are *CMMi* and *ISO 15504*. However, these maturity models are too complex to be applied to a SME of software development; apply those maturity models is non viable due the amount of resources needed for its implementation.

2. Basic Maturity Model

An appropriate continuous improvement process for the size of a SME of software development could be achieved by:

- establish revisions with people with more experience
- review past mistakes and act to ensure that such errors do not happen again
- make training sessions to refactor the most critical processes
- hold periodic meetings
- establish an adequate policy for the **key process areas**:

- **Causal analysis and resolution** (to identify causes of defects and other problems and take action to prevent them from occurring in the future)

- **Configuration management** (to establish and maintain the integrity of work products using configuration identification, configuration control, configuration status accounting, and configuration audits)

- **Decision analysis and resolution** (to analyze possible decisions using a formal evaluation process that evaluates identified alternatives against established criteria)

- **Integrated project management** (to establish and manage the project and the involvement of the relevant stakeholders according to an integrated and defined process that is tailored from the organization's set of standard processes).

- **Measurement and analysis** (to develop and sustain a measurement capability that is used to support management information needs)

- **Organizational innovation and deployment** (to select and deploy incremental and innovative improvements that measurably improve the organization's processes and technologies. The improvements support the organization's quality and process performance objectives as derived from the organization's business objectives)

- **Organizational process definition** (to establish and maintain a usable set of organizational process assets and work environment standards)

- **Organizational process focus** (plan, implement, and deploy organizational process improvements based on a thorough understanding of the current strengths and weaknesses of the organization's processes and process assets)

- **Organizational process performance** (to establish and maintain a quantitative understanding of the performance of the organization's set of standard processes in support of quality and process-performance objectives, and to provide the process performance data, baselines, and models to quantitatively manage the organization's projects)

- **Organizational training** (to develop the skills and knowledge of people so they can perform their roles effectively and efficiently)

- **Product integration** (to assemble the product from the product components, ensure that the product, as integrated, functions properly, and deliver the product)

- **Project monitoring and control** (to provide an understanding of the project's progress so that appropriate corrective actions can be taken when the project's performance deviates significantly from the plan)

- **Project planning** (to establish and maintain plans that define project activities)

- **Process and product quality assurance** (to provide staff and management with objective insight into processes and associated work products)

- **Quantitative project management** (to quantitatively manage the project's defined process to achieve the project's established quality and process-performance objectives)

- **Requirements development** (to produce and analyze customer, product, and product component requirements)

- **Requirements management** (to manage the requirements of the project's products and product components and to identify inconsistencies between those requirements and the project's plans and work products)

- **Risk management** (to identify potential problems before they occur so that risk-handling activities can be planned and invoked as needed across the life of the product or project to mitigate adverse impacts on achieving objectives)

- **Supplier agreement management** (to manage the acquisition of products from suppliers)

- **Technical solution** (to design, develop, and implement solutions to requirements. Solutions, designs, and implementations encompass products, product components, and product-related lifecycle processes either singly or in combination as appropriate)

- **Validation** (to demonstrate that a product or product component fulfils its intended use when placed in its intended environment)

- **Verification** (to ensure that selected work products meet their specified requirements)

Guideline Application Tool

1. Description

The tool used to perform this guideline will depend of the issue type.

8.5.2. Corrective action

ISO 9001:2008 Standard

The organization shall take action to eliminate the cause of nonconformities in order to prevent recurrence.

Corrective actions shall be appropriate to the effects of the nonconformities encountered.

A documented procedure shall be established to define requirements for

a) reviewing nonconformities (including customer complaints),

b) determining the causes of nonconformities,

c) evaluating the need for action to ensure that nonconformities do not recur,

d) determining and implementing action needed,

e) records of the results of action taken (see 4.2.4), and

f) reviewing the effectiveness of the corrective action taken.

ISO 90003:2004 Guideline

Where corrective action directly affects the software products, configuration management may be invoked to manage the changes. Management should review corrective actions that involve changes to the software life cycle processes. An organization's procedures for corrective action should take into account the requirement to prevent recurrence.

NOTE For further information, see ISO/IEC 12207:1995[11], 6.8, and ISO/IEC 12207:1995/Amd.1:2002[12], F.2.8 (problem resolution process).

Guideline Application

1. Description

Corrective actions must be done during all the software development process (for example, creating a bug). The corrective actions taken will vary depending of the type of nonconformity type.

Guideline Application Tool

1. Description

Basically, the tool used will be *Jira* and *Subversion*.

8.5.3. Preventive action

ISO 9001:2008 Standard

The organization shall determine action to eliminate the causes of potential nonconformities in order to prevent their occurrence. Preventive actions shall be appropriate to the effects of the potential problems.

A documented procedure shall be established to define requirements for

a) determining potential nonconformities and their causes,

b) evaluating the need for action to prevent occurrence of nonconformities,

c) determining and implementing action needed,

d) records of results of action taken (see 4.2.4), and

e) reviewing preventive action taken.

ISO 90003:2004 Guideline

Process assessment may be useful in the gathering of data to anticipate problems (see 8.2.3).

NOTE For further general guidance related to ISO 9001:2000, 8.5, see the following:

— ISO/IEC 12207:1995[11], 7.3.2 (process assessment), and ISO/IEC 12207:1995/Amd.1:2002[12], F.3.3.2 (process assessment);

— ISO/IEC 15504-2[23].

Guideline Application

1. Description

Several actions to eliminate the causes of potential nonconformities are the next:

- Update relevant procedures. Changes may refer to a spectrum of procedures, from those related to specific stages of software development or maintenance (e.g., changes in software

components, changes of contract review procedures in clauses dealing with proposals for small projects) to procedures of a general nature (e.g. changes of employee recruitment process, changes of the maximum or minimum number of participants in a formal design review).

• Changes in practices, including updating of relevant work instructions or update the existing procedures.

• Shifting to a development tool that is more effective and less prone to the detected faults.

• Improve of reporting methods, including changes in report content, frequency of reporting and reporting tasks. This direction is expected to improve prospects for identification of software system faults and their earlier detection, both resulting in substantial reductions in damages.

• Initiatives for training, retraining or updating staff. This direction is taken only in cases when the same training deficiencies are found in several teams.

• Improve the SQA infrastructure tools, in order to automatically create metrics of software nonconformities.

Guideline Application Tool

1. Description

Some of these software nonconformities can be detected with metric reports. In section *7.3.3. Design and development outputs* I have described some reports created by *Maven*. The project team leader will be responsible to read those reports and take the necessary corrective actions.

CHAPTER IV

IV – BOOKING BOOKS APPLICATION

1. Booking Books Overview

Booking Books, is an online library accessible by typing www.bookingbooks.com in a web browser; in this web page, the users can perform searches in the *Booking Books* catalogue, and rent some copies for all the available information resources. This web page will be a resource for learning, teaching, research and activities related to the operation and management of the knowledge as a whole. Its mission is to facilitate access and dissemination of information resources.

Once the users access to rent some books, these books are received at their home; and finally when read, the only thing they must do is bring the books to a mailbox. The main different between classic library and this one, is the same as the different between a classic bookstore and www.Amazon.com; in the first one you have to go physically to the store, and in the other the book is send to your house or office by postal mail.

I thought in this example, because many people go to his University or high school Library to rent some books, so they know how this management model works.

2. Booking Books Features

• The resources from the Library consist of all funds and bibliographic documentary, whatever its material and whatever their provenance: books, magazines, newspaper or others.

• Application should manage the collection of knowledge resources and perform the procurement of manage the technical inclusion, update or deletion of the information resource and maintaining a list of resources updated and accessible.

• The application should be able to operate via Web, configured as a unitary and centralized system.

• This application will have two kinds of users: a CRM (Customer Relationship Management) user who should be able to add, delete or modify the catalogue of books and related resources of information and manage the final users; and the final users, who will be able to rent them.

• Application will maintain a log of users that include personal data and contacts as well as a history of incidents.

• Application will ensure the confidentiality of all personal data in accordance with the provisions of the Organic Law of Data Protection and other regulations that develop.

• Final customers should be able to make suggestions for the better operation of the Service.

• Final customers will access to Booking Books application to rent some books; then these books will be received at their home; and finally, they mail the books again to the Booking Books postal address. Application should manage all these workflow.

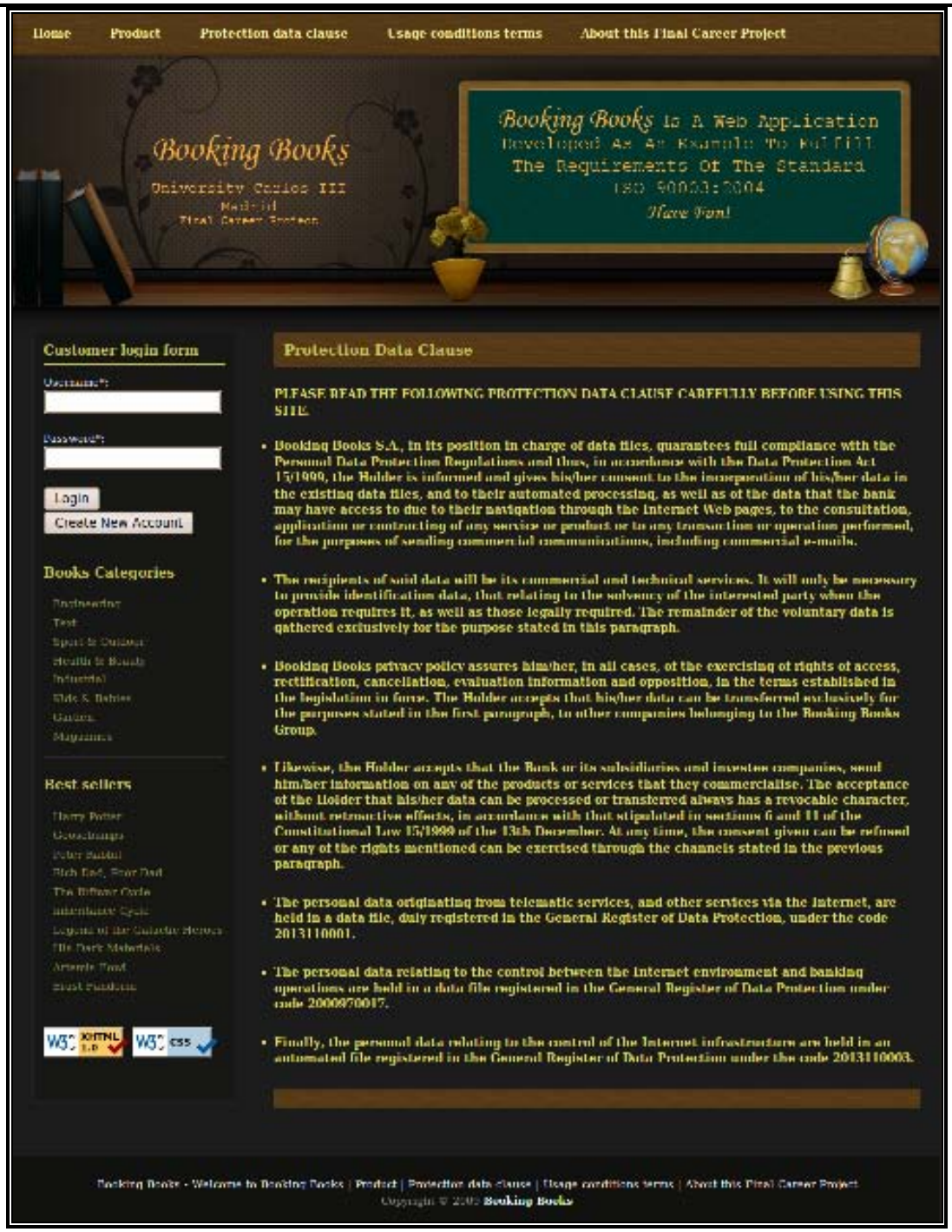
• The library in its service charter will set the deadline for responding to complaints and suggestions put forward by the users.

• Each service may be regulated by specific instructions that under no circumstances contravene the provisions of the international applicable regulation.

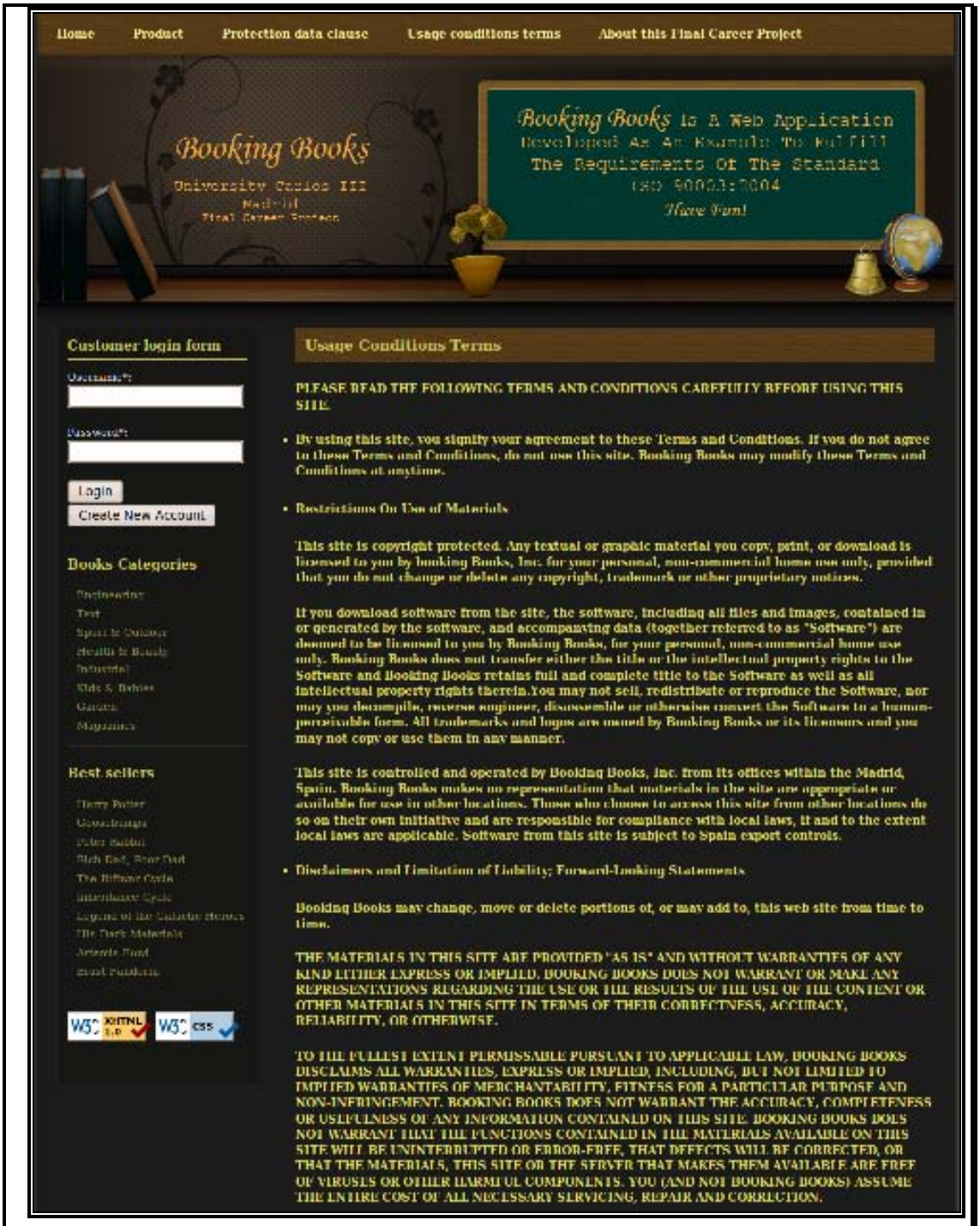
3. Application Snapshots

Picture IV-1: BookingBooks Application - Main Page

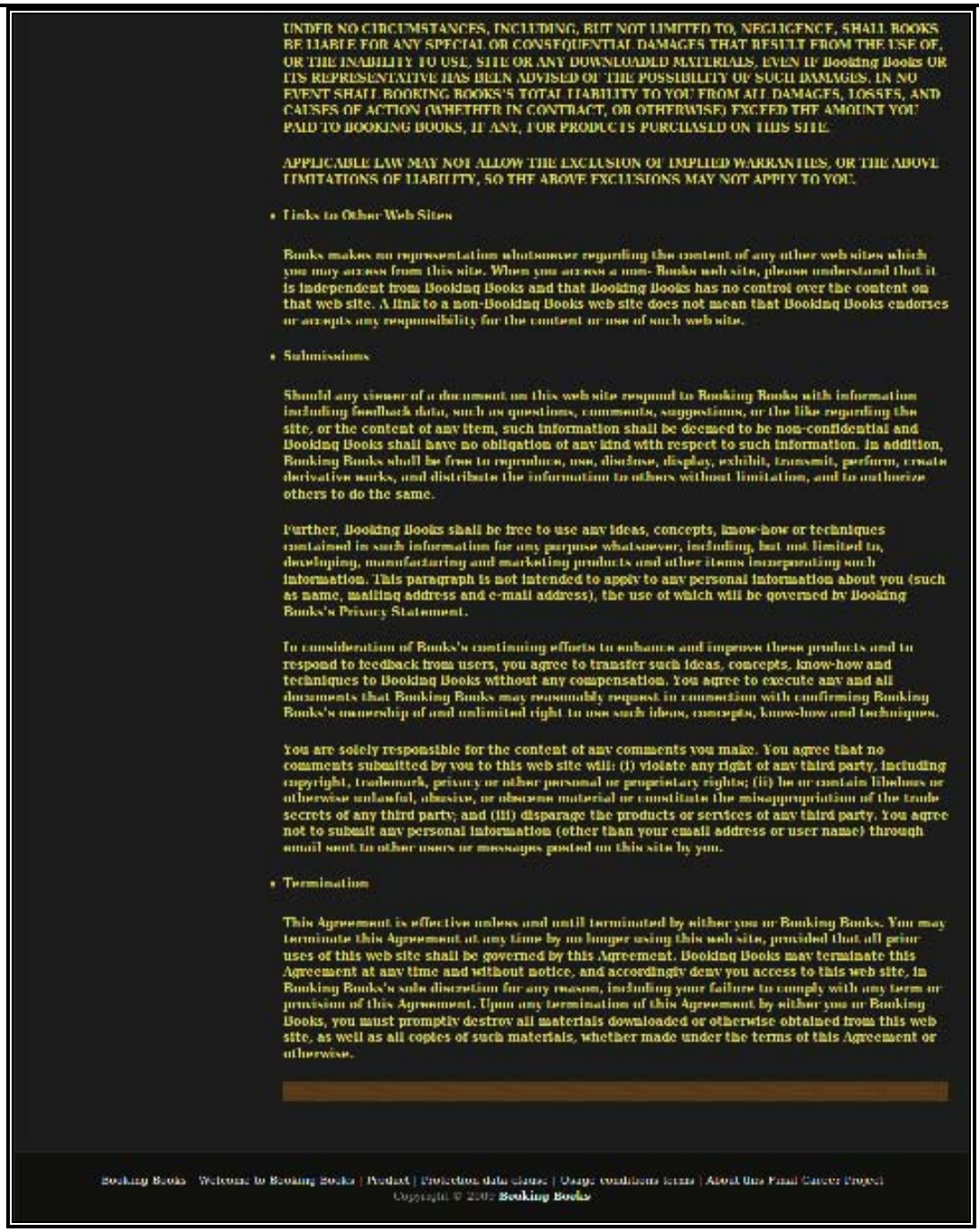
Picture IV-2: BookingBooks Application - Products



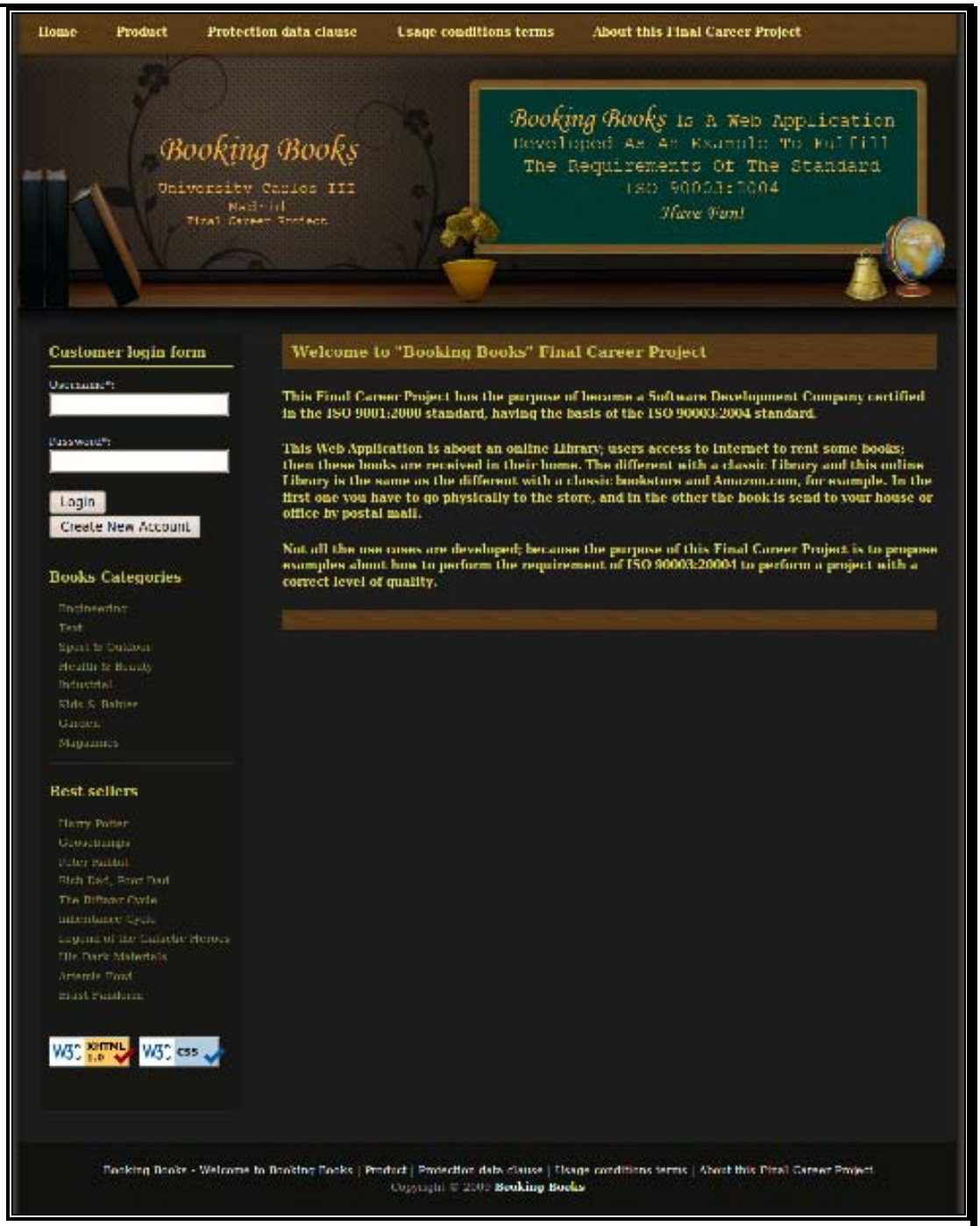
Picture IV-3: BookingBooks Application – Protection Data Clause



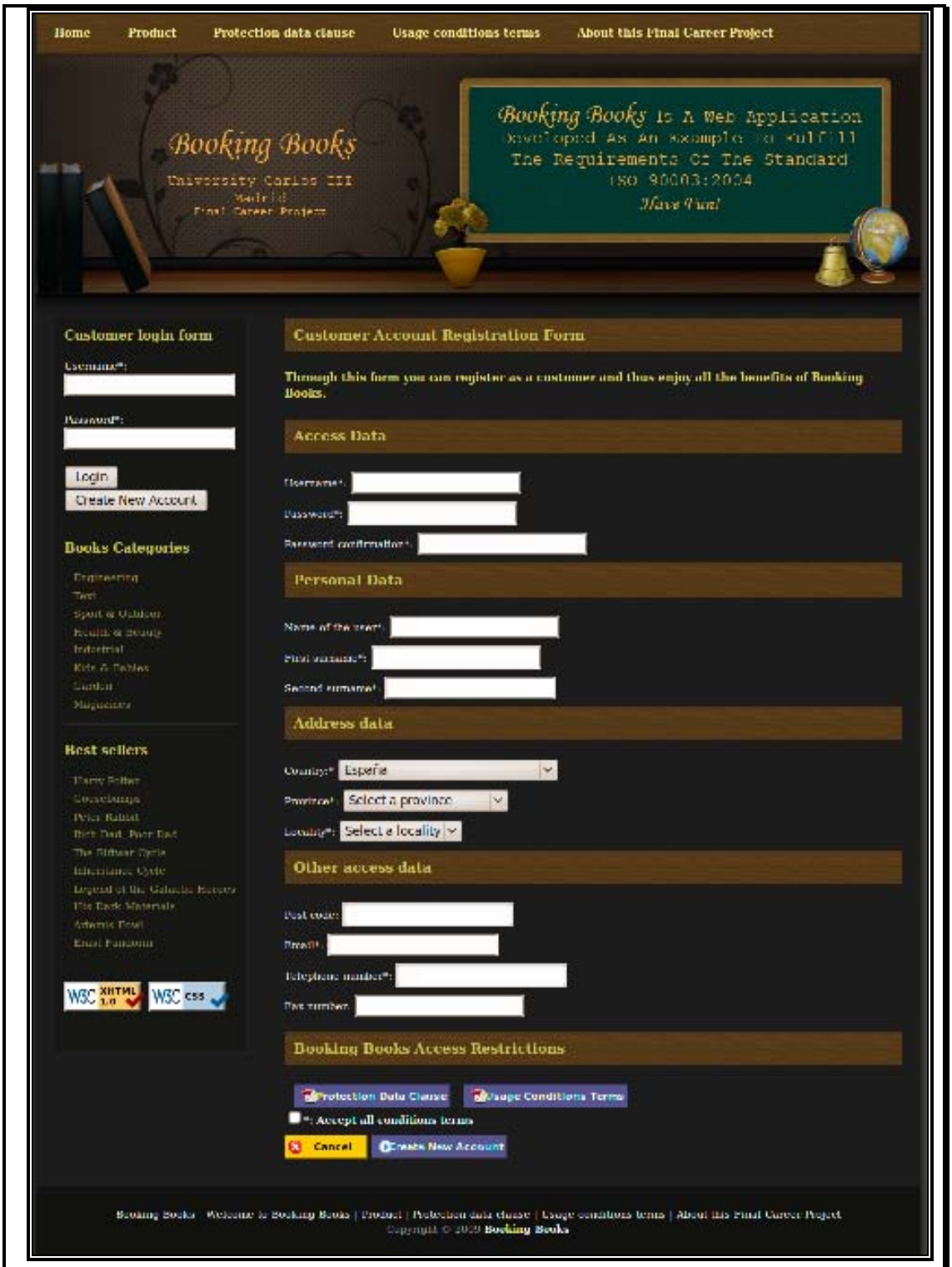
Picture IV-4: BookingBooks Application – Usage Conditions Terms(I)



Picture IV-5: BookingBooks Application – Usage Conditions Terms (II)

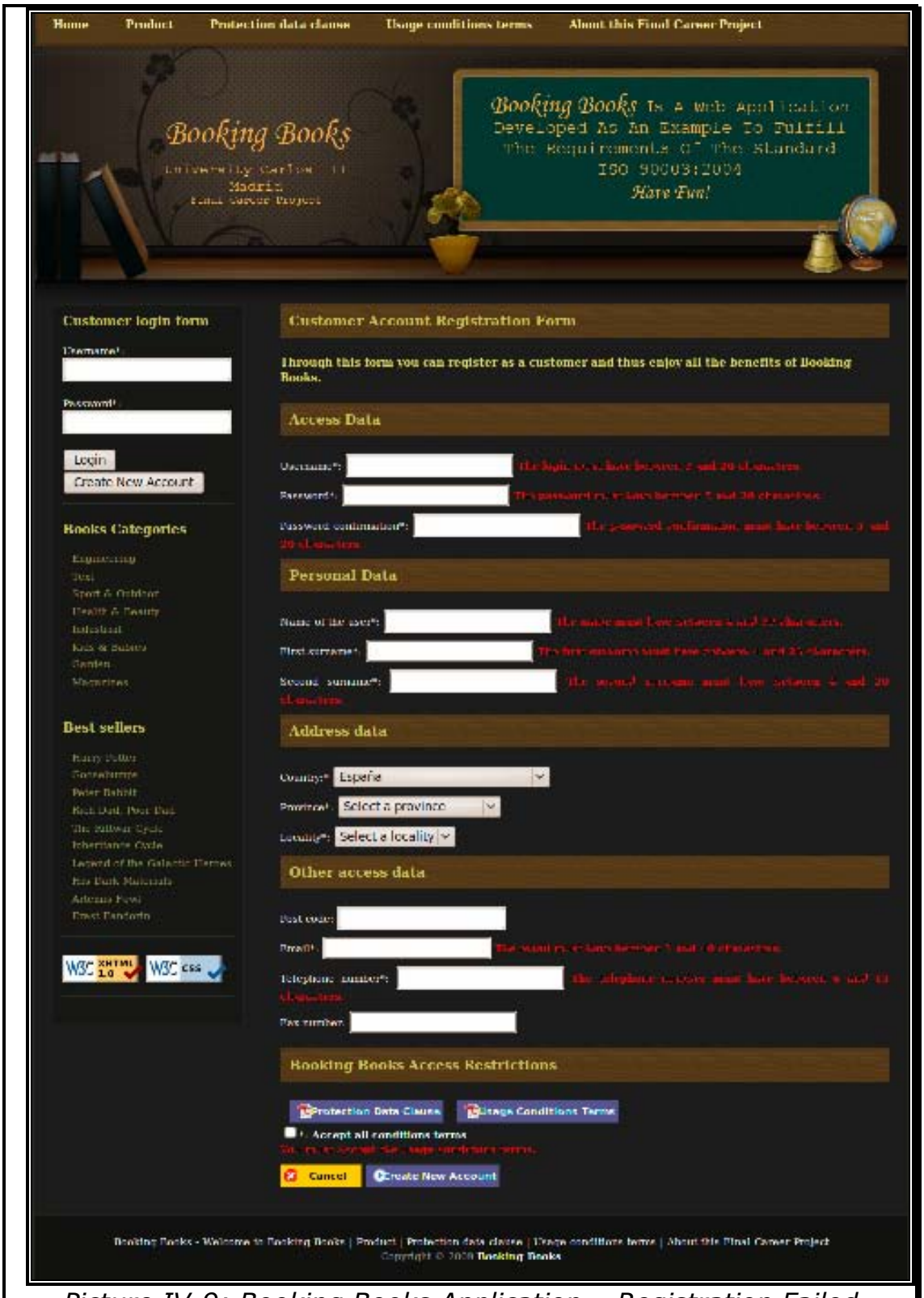


Picture IV-6: Booking Books Application - About



Picture IV-7: Booking Books Application – Create New Account

Picture IV-8: BookingBooks Application – Registration Success



Picture IV-9: Booking Books Application – Registration Failed

Picture IV-10: Booking Books Application – Login Success

Picture IV-11: Booking Books Application – Login Failed

CHAPTER V

V - CONCLUSIONS

1. Conclusion

1.1. Personal Reflection

The project has achieved the goals originally proposed.

It has helped to identify and highlight the main points to perform a successful implementation of a quality management system in an SME of software development, giving an example of how to follow the software development from requirements to the final delivery.

Among the most important points that every company must adapt to their circumstances are:

- identify what are the real issues that affect software development.
- the operational processes must be related to the daily work and not be a bureaucratic obstacle.
- involve the users in the process of implementing the quality management system so they know what to expect and what not to expect, defining the economic benefits, employment, and of any other type, so that people know how they are going to be benefit in particular.

To be successful, a quality management system must take into account all the employees of the company, and the needs of the organization. All the employees of the company have to be convinced that the measures taken are fair and necessary, otherwise, they will not follow the quality management system processes, and therefore the investment will be useless expenditure.

As defined, I think I got a very valid approach to the software development with a high level of quality, and I hope that readers will apply some of these ideas to their software development in the future.

1.2. Achieved Goals

By undertaking this project, I have achieved the next objectives:

- Understand and analyze the philosophy, key points and main issues of a quality management system.
- Offer a guide and the necessary tools to carry out the main issues of a quality management system.

• Create a solid and deep knowledge base of the J2EE technology, providing the basis for create application components for Web development.

• Used some different architectural patterns (MVC, DAO, DTO) to properly separate architecture responsibilities.

• Regarding to the design, use different design patterns (such as the Facade pattern), solving typical design errors.

• Regarding to the used tools, the main features are the next:

- All of them are free software (but Jira), thereby limiting the total cost ownership that is facing a company.

- I have used the Java programming because it is the commonly and most used; it is simple, object oriented, robust and secure, architecture neutral and portable, high performance and interpreted.

• Review not only the ISO 90003:2004 standard itself, but also ISO 9000:2005, and 9001:2008, explaining the differences between them and the reasons that led to their use.

• Writte this project in English, because English is the most important and used language for software development, allowing a bigger audience.

1.3. Future Work Lines

Once implemented all processes of a quality management system, the next step would be to obtain the ISO 9001 certification, and even implement a software process maturity model (CMMI, ISO 15504), to carry out continuous improvement on themselves.

2. Referenced Resources

2.1 Referenced Books

• **Better Builds with Maven.** How-to guide for Maven 2.0. Ed. Devzuz, April 2007.

• **Calidad de sistemas informáticos.** Mario Piattini; Felix O. García; Ismael Caballera Muñoz-Reja.

• **Calidad en el desarrollo y mantenimiento del software.** Mario G. Peattini; Félix O. García.

• **CMMI for Development (CMMI-DEV).** Carnegie Mellon Software Engineering Institute.

• **Ingeniería Del Software. Un enfoque práctico.** Roger S. Pressman, sexta edición.

• **Configuration Management Principles and Practice.** Anne Mette Jonassen Hass, 2002.

• **Expert One-on-One J2EE Design and Development.** Rod Johnson.

• **Head First Design Patterns.** Eric Freeman & Elisabeth Freeman. Ed O´Reilly.

• **Head First on Servlets & JSP.** Bryan Basham, Kathy Sierra and Bert Bates. Ed O´Reilly.

• **ISO 9000. Manual de Sistemas de Calidad.** David Hoyle. Ed. Paraninfo.

• **ISO 9000:2005. Quality management systems. Fundamentals and vocabulary.** ISO Standards.

• **ISO 9001:2008. Quality management systems. Requirements.** ISO Standards.

• **ISO 9001:2008. Quality management systems. Requirements.** ISO Standards.

• **ISO 9004:2009. Managing for the sustained success of an organization - A quality management approach.** ISO Standards.

• **ISO 90003:2004. Guidelines for the application of ISO 9001:2008 to computer software.** ISO Standards.

• **Java Persistente with Hibernate.** Christian Bauer, Gavin King. Ed. Manning.

• **Juran y la planificación para la calidad.** J. M. Juran.

• **Juran y el liderazgo para la calidad. Un manual para directivos.** Ed. Díaz De Santos.

• **Managing Web Projects.** Edward B. Farkas.

• **Manual de control de calidad.** J. M. Juran y Frank M. Gryna.

• **Mastering the Requirements Process, second edition.** Suzanne and James Robertson.

• **Norma ISO 9126-1- Software engineering — Product quality.** ISO Standards.

• **Object-Oriented Software Construction.** Bertrand Meyer.

• **Software Configuration Management Patterns: Effective Teamwork, Practical Integration.** Steve Berczuk, Brad Appleton, 2002.

• **Software Engineering.** 7th Edition, Ian Sommerville.

• **Software Quality Assurance: From Theory to Implementation.** Daniel Galin.

• **Técnicas para la gestión de la calidad.** Albert Badia y Sergio Bellido. Ed. Tecnos.

• **The Unified Modeling Language User Guide.** Booch, Rumbaugh, Jacobson, 2005.

• **The Unified Software Development Process.** Booch, Rumbaugh, Jacobson.

• **Writing Effective Use Cases.** Alistair Cockburn, 2000.

• **Struts in Action. Building web applications with the leading Java framework.** Ted Husted, Cedric Dumoulin, George Franciscus, David Winterfeldt. Ed Manning.

2.2 Referenced Papers

• **Reglas de Diseño e Implementación desde Casos de Uso.** David Lopez, 2006.

• **Code Conventions for the Java Programming Language.** 1997, Sun Microsystems, Inc.

2.3 Referenced Online Resources

• http://continuum.apache.org/

• http://easymock.org/

• http://en.wikipedia.org/

• http://jakarta.apache.org/jmeter/

• http://java.sun.com/

• http://java.sun.com/products/jsp/

• http://jboss.org/jbossas/

• http://junit.sourceforge.net/.

• http://maven.apache.org/

• http://subversion.apache.org/

• http://twiki.org/

• http://www.atlassian.com/software/jira/

• http://www.eclipse.org/

• http://www.hibernate.org/

• http://www.iso.org/

• http://www.pgadmin.org/

• http://www.postgresql.org/

• http://www.sei.cmu.edu/cmmi/

• http://www.springsource.org/

• http://www.ubuntu.com/

• http://www.volere.co.uk/

- http://xuse.sourceforge.net/
- http://www.zimbra.com/

ANNEX I: SOURCE CODE

1. Model Source Code (MVC Architecture)

1.1. CustomerAccount.java (Entity)

```
	/**
	 * Entity to persist the Customer Account data
	 */

	// Named Query to count the existing CustomerAccount accounts in database for an
	// email address
	@NamedQueries( {
		@NamedQuery(name = "countAllCustomersByEmailAddress", query = "select
count(distinct ca.id) "
			+ " from CustomerAccount ca where " + " ca.email like :email "),
		@NamedQuery(name = "retrieveCustomerByUsername", query = "select ca "
			+ " from CustomerAccount ca where "
			+ " ca.userName like :userName ") })
	// Hibernate ORM Annotation to persist as an Entity
	@Entity
	public class CustomerAccount {

	   // Query statement used in CustomerAccountDAO.java
	   public static final String QUERY_COUNT_CUSTOMERS_BY_EMAIL_ADDRESS =
"countAllCustomersByEmailAddress";
	   public static final String QUERY_RETRIEVE_CUSTOMER_BY_USERNAME =
"retrieveCustomerByUsername";

	   public static final String FIELD_ID_NAME = "id";
	   public static final String EMAIL_FIELD = "email";
	   public static final String USERNAME_BUSSINESS_RULE_KEY = "userName";

	   // Default constructor for Hibernate
	   CustomerAccount() {
	   }

	   // Public constructor with all the mandatory fields
	   public CustomerAccount(String username, String password, String name,
	         String firstSurname, String secondSurname, String email,
	         Location country, Location province, Location locality,
	         String telephoneNumber) {
	      this.userName = username;
	      this.password = password;
	      this.name = name;
	      this.firstSurname = firstSurname;
	      this.secondSurname = secondSurname;
	      this.email = email;
```

```
        this.country = country;
        this.province = province;
        this.locality = locality;
        this.telephoneNumer = telephoneNumber;
    }

    // Auto-generated Id value needed by Hibernate
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    // Login of the customer: business key (must be unique)
    @Column(unique = true)
    // User Name of the customer
    private String userName;
    // Name of the customer

    // Name of the customer
    @Column
    private String name;

    // Password of the customer
    @Column
    private String password;

    // First surname of the customer
    @Column
    private String firstSurname;
    // Second surname of the customer

    @Column
    private String secondSurname;

    // E-mail of the customer
    @Column
    private String email;

    // Post Code of the customer
    @Column
    private String postCode;

    // Country of the customer
    @OneToOne(fetch = FetchType.LAZY)
    private Location country;

    // Province of the customer Relation one to one with a Location Entity
    @OneToOne(fetch = FetchType.LAZY)
    private Location province;

    // Locality of the customer. Relation one to one with a Location Entity
    @OneToOne(fetch = FetchType.LAZY)
    private Location locality;

    // Number of telephone of the customer
    @Column
    private String telephoneNumer;

    // Number of fax of the customer
    @Column
    private String faxNumber;

    /*
     * (non-Javadoc)
```

```
     *
     * @see java.lang.Object#hashCode()
     */
    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result
                + ((userName == null) ? 0 : userName.hashCode());
        return result;
    }

    /*
     * Equals method
     *
     * @see java.lang.Object#equals(java.lang.Object)
     */
    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        CustomerAccount other = (CustomerAccount) obj;
        if (userName == null) {
            if (other.userName != null)
                return false;
        } else if (!userName.equals(other.userName))
            return false;
        return true;
    }

    /**
     * @return the id
     */
    public Long getId() {
        return id;
    }

    /**
     * @param id
     *            the id to set
     */
    public void setId(Long id) {
        this.id = id;
    }

    /**
     * @return the userName
     */
    public String getUserName() {
        return userName;
    }

    /**
     * @param userName
     *            the userName to set
     */
    public void setUserName(String userName) {
        this.userName = userName;
    }
```

```
/**
 * @return the name
 */
public String getName() {
   return name;
}

/**
 * @param name
 *          the name to set
 */
public void setName(String name) {
   this.name = name;
}

public String getPassword() {
   return password;
}

public void setPassword(String password) {
   this.password = password;
}

/**
 * @return the firstSurname
 */
public String getFirstSurname() {
   return firstSurname;
}

/**
 * @param firstSurname
 *          the firstSurname to set
 */
public void setFirstSurname(String firstSurname) {
   this.firstSurname = firstSurname;
}

/**
 * @return the secondSurname
 */
public String getSecondSurname() {
   return secondSurname;
}

/**
 * @param secondSurname
 *          the secondSurname to set
 */
public void setSecondSurname(String secondSurname) {
   this.secondSurname = secondSurname;
}

/**
 * @return the email
 */
public String getEmail() {
   return email;
}

/**
 * @param email
```

```
     *          the email to set
     */
    public void setEmail(String email) {
        this.email = email;
    }

    /**
     * @return the postCode
     */
    public String getPostCode() {
        return postCode;
    }

    /**
     * @param postCode
     *          the postCode to set
     */
    public void setPostCode(String postCode) {
        this.postCode = postCode;
    }

    /**
     * @return the country
     */
    public Location getCountry() {
        return country;
    }

    /**
     * @param country
     *          the country to set
     */
    public void setCountry(Location country) {
        this.country = country;
    }

    /**
     * @return the province
     */
    public Location getProvince() {
        return province;
    }

    /**
     * @param province
     *          the province to set
     */
    public void setProvince(Location province) {
        this.province = province;
    }

    /**
     * @return the locality
     */
    public Location getLocality() {
        return locality;
    }

    /**
     * @param locality
     *          the locality to set
     */
    public void setLocality(Location locality) {
```

```
            this.locality = locality;
        }

        /**
         * @return the telephoneNumer
         */
        public String getTelephoneNumer() {
            return telephoneNumer;
        }

        /**
         * @param telephoneNumer
         *          the telephoneNumer to set
         */
        public void setTelephoneNumer(String telephoneNumer) {
            this.telephoneNumer = telephoneNumer;
        }

        /**
         * @return the faxNumber
         */
        public String getFaxNumber() {
            return faxNumber;
        }

        /**
         * @param faxNumber
         *          the faxNumber to set
         */
        public void setFaxNumber(String faxNumber) {
            this.faxNumber = faxNumber;
        }

    }
```

1.2. CustomerAccountDao.java (DAO)

```
    /**
     * Interface that all DAO implementations (EJB3 based, JPA based, etc...) must
     * implement.
     *
     * <p>
     * Only specific finder methods should be defined here. The rest of the methods
     * are already defined in the BaseDAO interface.
     * </p>
     *
     * <p>
     * The first BaseDAO template parameter is the class of the entity to be
     * persisted and the second template parameter is the business key type of the
     * entity to be persisted.
     * </p>
     */
    public interface CustomerAccountDao extends BaseDAO<CustomerAccount, String>
{

        /**
         * Method to know if a CustomerAccount email address exist or not
         *
         * @param emailAddres
```

```
     *          the email address to search
     * @return true or false if CustomerAccount already exist or not
     */
    public Boolean checkCustomerAccountEmailAddressAlreadyExist(
            final String emailAddres);

    /**
     * Return the CustomerAccount, or an exception otherwise
     *
     * @param userName
     *          the userName to search
     * @return the CustomerAccount found for the userName
     */
    public CustomerAccount loadByUsername(String userName)
            throws NoSuchEntityException, DaoException;

}
```

1.3. CustomerAccountDaoJpaImpl.java (DAO Impl)

```
/**
 * DAO performing all persistence related operations.
 */

public class CustomerAccountDaoJpaImpl extends
        AbstractBaseJpaDao<CustomerAccount, String> implements
        CustomerAccountDao {

    /**
     * {@inheritDoc}
     */
    public List<CustomerAccount> findAll() throws DaoException {
        return super.findAll(CustomerAccount.class);
    }

    /**
     * {@inheritDoc}
     */

    public CustomerAccount loadByUsername(String userName)
            throws NoSuchEntityException, DaoException {

        EntityManager em = super.getEntityManager();

        try {
            return (CustomerAccount) em.createNamedQuery(
                    CustomerAccount.QUERY_RETRIEVE_CUSTOMER_BY_USERNAME)
                    .setParameter(CustomerAccount.USERNAME_BUSSINESS_RULE_KEY,
                            userName).getSingleResult();
        } catch (NoResultException e) {
            throw new NoSuchEntityException(CustomerAccount.class);
        }
    }

    /**
     * {@inheritDoc}
     */
```

```
    public Boolean checkCustomerAccountEmailAddressAlreadyExist(
            final String email) {
        Long xx = (Long) getEntityManager().createNamedQuery(
                CustomerAccount.QUERY_COUNT_CUSTOMERS_BY_EMAIL_ADDRESS)
                .setParameter(CustomerAccount.EMAIL_FIELD, "%" + email + "%")
                .getSingleResult();
        if (xx > 0) {
            return true;
        } else {
            return false;
        }
    }

    public CustomerAccount load(String id) throws NoSuchEntityException,
            DaoException {
        return super.loadByUniqueField(CustomerAccount.class,
                CustomerAccount.FIELD_ID_NAME, id);
    }
}
```

1.4. CustomerAccountManager.java (Manager)

```
/**
 * This is the core's entry point to manage Customer Accounts.
 */
public interface CustomerAccountManager {

    /**
     * Obtains all the CustomerAccount stored in the repository.
     *
     * @return A <code>List</code> of <code>CustomerAccount</code>.
     *
     */
    List<CustomerAccount> getAllCustomerAccounts();

    /**
     * Persist a new CustomerAccount
     *
     * @param customerAccount
     * @return
     * @throws CustomerAccountUsernameAlreadyExistException
     *             if the userName already exist
     * @throws CustomerAccountEmailAddressAlreadyExistException
     *             if the email already exist
     */
    CustomerAccount create(CustomerAccount customerAccount)
            throws CustomerAccountUsernameAlreadyExistException,
            CustomerAccountEmailAddressAlreadyExistException;

    /**
     * Method to load a CustomerAccount, or throw an exception otherwise
     *
     * @param userName
     *          the userName to search by
     * @return CustomerAccount if the userName already exist
     */
    public CustomerAccount loadByUsername(String userName); }
```

1.5. CustomerAccountManagerImpl.java (Manager Impl)

```
      /**
       * Implementation of the <code>CustomerAccountManager</code>.
       */
      public class CustomerAccountManagerImpl implements CustomerAccountManager {

         private CustomerAccountDao customerAccountDao;

         /**
          * {@inheritDoc}
          */
         public List<CustomerAccount> getAllCustomerAccounts() {
            try {
               return customerAccountDao.findAll();
            } catch (DaoException e) {
               throw    new    CustomerAccountManagerRuntimeException(e.getMessage(),
e);
            }
         }

         /**
          * {@inheritDoc}
          */
         public CustomerAccount create(CustomerAccount customerAccount)
               throws CustomerAccountUsernameAlreadyExistException,
               CustomerAccountEmailAddressAlreadyExistException {

            try {
               customerAccountDao.loadByUsername(customerAccount.getUserName());
               throw new CustomerAccountUsernameAlreadyExistException(
                     "Customer Account with userName = "
                           + customerAccount.getUserName()
                           + " already exists in the system.");
            } catch (NoSuchEntityException e) {
            }

            if (customerAccountDao
                  .checkCustomerAccountEmailAddressAlreadyExist(customerAccount
                        .getEmail()) == true)
               throw new CustomerAccountEmailAddressAlreadyExistException(
                     "Customer Account with email = "
                           + customerAccount.getEmail()
                           + " already exists in the system.");

            try {
               CustomerAccount createdCustomerAcccount = customerAccountDao
                     .create(customerAccount);
               return createdCustomerAcccount;
            } catch (InvalidEntityException e) {
               throw new CustomerAccountManagerRuntimeException(
                     "The Customer Account is invalid: " + e.getMessage(), e);
            } catch (DuplicatedEntityException e) {
               throw new CustomerAccountManagerRuntimeException(
                     "The Customer Account is invalid: " + e.getMessage(), e);
            }
         }
```

```
        /**
         * {@inheritDoc}
         */
        public CustomerAccount loadByUsername(String userName) {
           try {
              return customerAccountDao.loadByUsername(userName);
           } catch (DaoException e) {
              throw new CustomerAccountManagerRuntimeException("DaoException in"
                    + CustomerAccountManagerImpl.class, e);
           } catch (NoSuchEntityException e) {
              throw new CustomerAccountManagerRuntimeException("DaoException in"
                    + CustomerAccountManagerImpl.class, e);
           }
        }

        // ..... Dependency Injection ...
        /**
         *
         * @param customerAccountDao
         */

        public void setCustomerAccountDao(CustomerAccountDao customerAccountDao)
{
           this.customerAccountDao = customerAccountDao;
        }

     }
```

2. View Source Code (MVC Architecture)

2.1. CreateNewCustomerAccount.jsp

```
     <%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8" %>
     <%@ taglib prefix="html" uri="http://struts.apache.org/tags-html" %>
     <%@ taglib prefix="logic" uri="http://struts.apache.org/tags-logic" %>
     <%@ taglib prefix="bean" uri="http://struts.apache.org/tags-bean" %>
     <%@                                                                    page
import="com.almiralabs.bookingbooks.ui.controller.ManageCustomersActionForm" %>

     <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
       <script type="text/javascript" language="Javascript"
             src="<html:rewrite page='/js/geographicSelects.js'/>">
       </script>
        <script type="text/javascript">

           // display = "none"/"inline" to force a realignment of the page layout.
           // if you don't want this just use: = iState ? "visible" : "hidden";
           function toggleBox(szDivID, iState) // 1 visible, 0 hidden
           {
              if (document.layers)      //NN4+
              {
                 document.layers[szDivID].display = iState ? "inline" : "none";
              }
```

```
            else if (document.getElementById)      //gecko(NN6) + IE 5+
            {
               var obj = document.getElementById(szDivID);
               obj.style.display = iState ? "inline" : "none";
            }
            else if (document.all)    // IE 4
            {
               document.all[szDivID].style.display = iState ? "inline" : "none";
            }
         }
      </script>

      <div id="content_right">
      <h1><bean:message key="customer.registration.form.tile"/></h1>
      <h2><bean:message key="customer.registration.form.description"/></h2>

      <html:messages id="saveOk" property="saveOk" message="true">
         <blockquote class="Info">
            <bean:write name="saveOk"/>
         </blockquote>
      </html:messages>
      <span class="Info"><html:messages id="saveOk"/></span>

     <html:form action="/createNewCustomerAccount" method = "POST">
      <input type="hidden" name="operation" value="create"/>

      <h1><bean:message key="customer.registration.form.access.data"/></h1>

      <p class="W750" style="padding-top : 0px;padding-left : 0px;padding-right : 0px;padding-bottom : 0px;margin-top : 0px;margin-left : 0px;margin-right : 0px;margin-bottom : 0px;">
      <p>
        <bean:message
key="customer.registration.form.username"/><span>*</span>:
         <html:text name="manageCustomersActionForm"
                        maxlength="50"
                        property="userName"/>
         <span class="error"><html:errors property="userName"/></span>
      </p>

      <p>
         <bean:message
              key="customer.registration.form.password"/><span>*</span>:

         <html:password name="manageCustomersActionForm"
                        maxlength="50"
                        property="password"/>

         <span class="error"><html:errors property="password"/></span>
      </p>

      <p>
         <bean:message

key="customer.registration.form.password.confirmation"/><span>*</span>:

         <html:password name="manageCustomersActionForm"
                        maxlength="50"
                        property="passwordConfirmation"/>
         <span                                          class="error"><html:errors property="passwordConfirmation"/></span>
      </p>
     </p>
```

```
      <h1><bean:message key="customer.registration.form.personal.data"/></h1>

         <p class="W750" style="padding-top : 0px;padding-left : 0px;padding-right :
0px;padding-bottom : 0px;margin-top : 0px;margin-left : 0px;margin-right :
0px;margin-bottom : 0px;">
         <p>
            <bean:message key="customer.registration.form.name"/><span>*</span>:
            <html:text name="manageCustomersActionForm"
                              maxlength="50"
                              property="name"/>
            <span class="error"><html:errors property="name"/></span>
         </p>

         <p>
            <bean:message
                 key="customer.registration.form.first.surname"/><span>*</span>:

            <html:text name="manageCustomersActionForm"
                              maxlength="50"
                              property="firstSurname"/>
            <span class="error"><html:errors property="firstSurname"/></span>
         </p>

         <p>
            <bean:message
                 key="customer.registration.form.second.surname"/><span>*</span>:

            <html:text name="manageCustomersActionForm"
                              maxlength="50"
                              property="secondSurname"/>
            <span class="error"><html:errors property="secondSurname"/></span>
         </p>
         </p>

         <h1><bean:message key="customer.registration.form.address.data"/></h1>

         <p class="W750" style="padding-top : 0px;padding-left : 0px;padding-right :
0px;padding-bottom : 0px;margin-top : 0px;margin-left : 0px;margin-right :
0px;margin-bottom : 0px;">

      <!-- pais -->
         <p>
            <bean:message
key="customer.registration.form.country"/>:<span>*</span>
            <html:select                styleId="country"                property="country"
onchange="resetList('province');resetList('town');LoadList('/bookingbooks/obtainGeograp
hicData.do?show=provByCountry&id=',this.value,'province');">
            <html:option                                              value="-1"><bean:message
key="customer.registration.form.select.country"/></html:option>
            <logic:present                              name="manageCustomersActionForm"
property="countryList">
               <logic:iterate    id="country"        name="manageCustomersActionForm"
property="countryList">
                  <bean:define   id="countryName"   name="country"   property="name"
type="java.lang.String"/>
                  <bean:define      id="countryId"      name="country"      property="id"
type="java.lang.Long"/>
                  <html:option value="<%=countryId.toString() %>"><%=countryName
%></html:option>
               </logic:iterate>
            </logic:present>
```

```
            </html:select>
            <span class="error"><html:errors property="country"/></span>
          </p>

       <!-- provincia -->
          <p>
            <bean:message
key="customer.registration.form.province"/><span>*</span>:
            <html:select styleId="province" property="province"
onchange="resetList('town');LoadList('/bookingbooks/obtainGeographicData.do?id=',this.
value,'town');">
            <html:option value="-1"><bean:message
key="customer.registration.form.select.province"/></html:option>
            <logic:present name="manageCustomersActionForm"
property="provinceList">
               <logic:iterate id="province" name="manageCustomersActionForm"
property="provinceList">
                  <bean:define id="provinceName" name="province" property="name"
type="java.lang.String"/>
                  <bean:define id="provinceId" name="province" property="id"
type="java.lang.Long"/>
                  <html:option value="<%=provinceId.toString()
%>"><%=provinceName %></html:option>
               </logic:iterate>
            </logic:present>
            </html:select>
            <span class="error"><html:errors property="province"/></span>
          </p>

       <!-- localidad -->
          <p>
            <bean:message
key="customer.registration.form.locality"/><span>*</span>:
            <html:select styleId="town" property="town">
            <html:option value="-1"><bean:message
key="customer.registration.form.select.locality"/></html:option>
            <logic:present name="manageCustomersActionForm"
property="localityList">
               <logic:iterate id="locality" name="manageCustomersActionForm"
property="localityList">
                  <bean:define id="localityName" name="locality" property="name"
type="java.lang.String"/>
                  <bean:define id="localityId" name="locality" property="id"
type="java.lang.Long"/>
                  <html:option value="<%=localityId.toString() %>"><%=localityName
%></html:option>
               </logic:iterate>
            </logic:present>
            </html:select>
            <span class="error"><html:errors property="town"/></span>
          </p>
       </p>

          <h1><bean:message
key="customer.registration.form.other.address.data"/></h1>
          <p class="W750" style="padding-top : 0px;padding-left : 0px;padding-right :
0px;padding-bottom : 0px;margin-top : 0px;margin-left : 0px;margin-right :
0px;margin-bottom : 0px;">

          <p>
            <bean:message
                  key="customer.registration.form.post.code"/>:
```

```
            <html:text name="manageCustomersActionForm"
                              maxlength="50"
                              property="postCode"/>
            <span class="error"><html:errors property="postCode"/></span>
          </p>

          <p>
            <bean:message
                 key="customer.registration.form.email"/><span>*</span>:

            <html:text name="manageCustomersActionForm"
                              maxlength="50"
                              property="email"/>
            <span class="error"><html:errors property="email"/></span>
          </p>

          <p>
            <bean:message

key="customer.registration.form.telephone.number"/><span>*</span>:

            <html:text name="manageCustomersActionForm"
                              maxlength="50"
                              property="telephoneNumber"/>
            <span class="error"><html:errors property="telephoneNumber"/></span>
          </p>

          <p>
            <bean:message
                 key="customer.registration.form.fax.number"/>:

            <html:text name="manageCustomersActionForm"
                              maxlength="50"
                              property="faxNumber"/>
            <span class="error"><html:errors property="faxNumber"/></span>
          </p>
         </p>

          <p>
          <h1><bean:message
key="customer.registration.form.access.restrictions"/></h1>
          <input       type="button"       class="BtnPdfAzul"       value="<bean:message
key="download.data.conditions.terms"/>"      onclick="document.location='<html:rewrite
page="<%="/toPdfServlet.pdf?operation=createProtectionData"%>"/>'"/>

          <input       type="button"       class="BtnPdfAzul"       value="<bean:message
key="download.terms.and.contitions.use"/>"  onclick="document.location='<html:rewrite
page="<%="/toPdfServlet.pdf?operation=createTermsAndConditions"%>"/>'"/>
          </p>
          <br/>
          <p>
            <html:checkbox  styleClass="NoStyle"  name="manageCustomersActionForm"
property="acceptAllUsageConditions"/><span>*</span>:
            <strong><bean:message
key="customer.registration.form.data.accept.all.conditions.terms"/></strong>
            <br/>
            <span                                                  class="error"><html:errors
property="acceptAllUsageConditions"/></span>
          </p>

             <input type="button" class = "BtnCruzAmarillo"
                  value="<bean:message
key="customer.registration.form.cancel.create.new.account.button.text"/>"
```

```
                    onclick="javascript:  window.location  =  '<html:rewrite
action="customermain.do"/>'"/>

                <html:submit styleClass="BtnFleAzulNoFixedWidth">
                   <bean:message
key="customer.registration.form.create.new.account.button.text"/>
                </html:submit>
          </html:form>

            </div>

            <!-- end of content right -->
            <div class="cleaner_with_height"> </div>
          </div>
          <!-- end of content -->
        </div>
        <!-- end of container -->
```

3. Controller Source Code (MVC Architecture)

3.1. ManageCustomersAction.java

```
        /**
         * This action groups all <code>CustomerAccount</code> related operations.
         */
        public class ManageCustomersAction extends DispatchAction {

            // Bean used in this Action
            public  static  final  String  BEAN_CUSTOMER_ACCOUNT_MANAGER  =
"customerAccountManager";

            // Error messages
            public  static  final  String  ERROR_USERNAME_ALREADY_TAKEN  =
"customer.account.username.already.taken";
            public  static  final  String  ERROR_EMAIL_ALREADY_TAKEN  =
"customer.account.email.address.alreay.taken";
            public static final String CUSTOMER_ACCOUNT_SUCCESSFULLY_CREATED =
"customer.account.successfully.created";

            // Forward to struts-config.xml file
            public static final String FORWARD_CREATE_NEW_CUSTOMER_ACCOUNT =
"preapareNewAccount";
            public static final String MAPPING_CREATE_SUCCESS = "success";
            public  static  final  String  FORWARD_USERNAME_ALREADY_TAKEN  =
"usernameAlreadyTaken";
            public  static  final  String  FORWARD_EMAIL_ALREADY_TAKEN  =
"emailAlreadyTaken";

            /**
             * Creates a new CustomerAccount
             *
             * @param mapping
             * @param form
             * @param request
             * @param response
```

```
         * @return
         * @throws Exception
         *           if an exception in the CustomerAccountManager happens
         */
        @SuppressWarnings("deprecation")
        public ActionForward create(ActionMapping mapping, ActionForm form,
              HttpServletRequest request, HttpServletResponse response)
              throws Exception {
           ActionErrors actionErrors = new ActionErrors();

           // Obtain an instance of the CustomerAccountManager
           WebApplicationContext ctx = WebApplicationContextUtils
                 .getRequiredWebApplicationContext(request.getSession()
                       .getServletContext());
           CustomerAccountManager customerAccountManager =
(CustomerAccountManager) ctx
                 .getBean(BEAN_CUSTOMER_ACCOUNT_MANAGER);

           ManageCustomersActionForm actionForm = (ManageCustomersActionForm)
form;

           // Create the CustomerAccount object with the data provided by the view

           CustomerAccount customerAccount = new CustomerAccount(
                 actionForm.getUserName(),
                 actionForm.getPassword(),
                 actionForm.getName(),
                 actionForm.getFirstSurname(),
                 actionForm.getSecondSurname(),
                 actionForm.getEmail(),
                 GeographicLocationUtils.findLocation(ctx, actionForm
                       .getCountry()),
                 GeographicLocationUtils.findLocation(ctx, actionForm
                       .getProvince()),
                 GeographicLocationUtils.findLocation(ctx, actionForm.getTown()),
                 actionForm.getTelephoneNumber());

           customerAccount.setPostCode(actionForm.getPostCode());
           customerAccount.setFaxNumber(actionForm.getFaxNumber());

           try {
              // Store the customerAccount in the repository
              customerAccountManager.create(customerAccount);
              ActionMessages message = new ActionMessages();
              message.add("saveOk", new ActionMessage(
                    CUSTOMER_ACCOUNT_SUCCESSFULLY_CREATED));
              saveMessages(request, message);

           } catch (CustomerAccountUsernameAlreadyExistException e) {
              // Show an error message if the userName is already taken
              actionErrors.add("userName", new ActionMessage(
                    ERROR_USERNAME_ALREADY_TAKEN));
              saveErrors(request, actionErrors);
              actionForm.setLocations(ctx);
              // There has been an error, redirect to the error page
              return mapping.findForward(FORWARD_USERNAME_ALREADY_TAKEN);
           } catch (CustomerAccountEmailAddressAlreadyExistException e) {
              // Show an error message if the email is already taken
              actionErrors.add("email", new ActionMessage(
                    ERROR_EMAIL_ALREADY_TAKEN));
              saveErrors(request, actionErrors);
              // There has been an error, redirect to the error page
              actionForm.setLocations(ctx);
```

```
                return mapping.findForward(FORWARD_EMAIL_ALREADY_TAKEN);
            }
            // No problem, forward to the sucess page
            return mapping.findForward(MAPPING_CREATE_SUCCESS);
        }

        /**
         * Show the ActionForm to create a new CustomerAccount
         *
         * @param mapping
         * @param form
         * @param request
         * @param response
         * @return
         */
        public ActionForward showFormNewCustomerAccount(ActionMapping mapping,
                ActionForm form, HttpServletRequest request,
                HttpServletResponse response) {
            // Forward to createNewCustomerAccount.jsp
            WebApplicationContext ctx = WebApplicationContextUtils
                    .getRequiredWebApplicationContext(request.getSession()
                            .getServletContext());
            ManageCustomersActionForm actionForm = (ManageCustomersActionForm)
form;

            actionForm.setCountryList(GeographicLocationUtils.getCountries(ctx));

            try {
                actionForm.setCountry(GeographicLocationUtils.getSpain(ctx).getId()
                        .toString());
            } catch (NoSuchEntityException e) {
            }
            updateProvinces(ctx, actionForm);

            return
mapping.findForward(FORWARD_CREATE_NEW_CUSTOMER_ACCOUNT);
        }

        /**
         * Update the list of provinces
         *
         * @param ctx
         * @param actionForm
         */
        private void updateProvinces(WebApplicationContext ctx,
                ManageCustomersActionForm actionForm) {

            if (actionForm.getCountry() != null
                    && actionForm.getCountry().length() > 0) {

                String countryStrId = actionForm.getCountry();

                actionForm.setProvinceList(GeographicLocationUtils
                        .getProvinces(GeographicLocationUtils.findLocation(ctx,
                                countryStrId)));
            }
        }
    }
```

3.2. ManageCustomersActionForm.java

```
	/**
	 * ActionForm to manage the <code>ManageCustomerAction</code> Struts action.
	 */
	@SuppressWarnings("serial")
	public class ManageCustomersActionForm extends ActionForm {

	   // Operation used to create a customer account
	   private static final String CREATE_CUSTOMER_ACCOUNT = "create";

	   // Length of every field of the CustomerAccount
	   private final int USERNAME_PASSWORD_MAX_LENGTH = 20;
	   private final int USERNAME_MIN_LENGTH = 3;
	   private final int PASSWORD_MIN_LENGTH = 5;

	   private final int NAME_FIRST_SURNAME_SECOND_SURNAME_MAX_LENGTH =
50;
	   private final int NAME_FIRST_SURNAME_SECOND_SURNAME_MIN_LENGTH = 4;
	   private final int EMAIL_MAX_LENGTH = 40;
	   private final int EMAIL_MIN_LENGTH = 7;
	   private final int POST_CODE_MAX_LENGTH = 10;
	   private final int TELEPHONE_NUMBER_MAX_LENGTH = 15;
	   private final int TELEPHONE_NUMBER_MIN_LENGTH = 6;
	   private final int FAX_NUMBER_MAX_LENGTH = 15;

	   // Regular expression for alphanumeric characters
	   Pattern ALPHANUMERIC_PATTERN = Pattern
	         .compile("\\A[0-9a-zA-Z áéíóúÁÉÍÓÚÇçñÑ\t]*\\Z");

	   // Message mapped to user-interface-messages.properties

	   private static final String INVALID_USERNAME_LENGTH =
"customer.creation.account.invalid.length.username";
	   private static final String INVALID_PASSWORD_LENGTH =
"customer.creation.account.invalid.length.password";
	   private static final String INVALID_PASSWORD_CONFIRMATION_LENGTH =
"customer.creation.account.invalid.length.password.confirmation";
	   private static final String PASSWORD_ARE_NOT_THE_SAME =
"customer.creation.account.password.are.not.the.same";
	   private static final String INVALID_NAME_LENGTH =
"customer.creation.account.invalid.length.name";
	   private static final String INVALID_NAME_FORMAT =
"customer.creation.account.invalid.format.name";
	   private static final String INVALID_FIRST_SURNAME_LENGTH =
"customer.creation.account.invalid.length.first.surname";
	   private static final String INVALID_FIRST_SURNAME_FORMAT =
"customer.creation.account.invalid.format.first.surname";
	   private static final String INVALID_SECOND_SURNAME_LENGTH =
"customer.creation.account.invalid.length.second.surname";
	   private static final String INVALID_SECOND_SURNAME_FORMAT =
"customer.creation.account.invalid.format.second.surname";
	   private static final String INVALID_EMAIL_LENGTH =
"customer.creation.account.invalid.length.email";
	   private static final String INVALID_EMAIL_FORMAT =
"customer.creation.account.invalid.format.email";
	   private static final String COUNTRY_NON_SELECTED =
"customer.creation.account.invalid.country.mandatory";
	   private static final String PROVINCE_NON_SELECTED =
"customer.creation.account.invalid.province.mandatory";
```

```
        private static final String LOCALITY_NON_SELECTED =
"customer.creation.account.invalid.town.mandatory";
        private static final String INVALID_POST_CODE_LENGTH =
"customer.creation.account.invalid.length.post.code";
        private static final String INVALID_POST_CODE_FORMAT =
"customer.creation.account.invalid.format.post.code";
        private static final String INVALID_TELEPHONE_NUMBER_LENGTH =
"customer.creation.account.invalid.length.telephone";
        private static final String INVALID_TELEPHONE_NUMBER_FORMAT =
"customer.creation.account.invalid.format.telephone";
        private static final String INVALID_FAX_NUMBER_LENGTH =
"customer.creation.account.invalid.length.fax";
        private static final String INVALID_FAX_NUMBER_FORMAT =
"customer.creation.account.invalid.format.fax";
        private static final String ACCEPT_USAGE_CONDITIONS_UNSELECTED =
"customer.creation.account.usage.conditions.not.selected";

        // Fields of the ActionForm
        private String userName;
        private String password;
        private String passwordConfirmation;
        private String name;
        private String firstSurname;
        private String secondSurname;
        private String email;
        private String postCode;
        private List<Location> countryList;
        private List<Node> provinceList;
        private List<Node> localityList;
        private String country;
        private String province;
        private String town;
        private String telephoneNumber;
        private String faxNumber;
        private boolean acceptAllUsageConditions = false;

        /**
         * @see ActionForm#validate(ActionMapping, javax.servlet.ServletRequest)
         */
        @Override
        public ActionErrors validate(ActionMapping mapping,
              HttpServletRequest request) {
           ActionErrors actionErrors = new ActionErrors();

           String operParam = mapping.getParameter();
           String operation = request.getParameter(operParam);

           // Validate when creating a new customer
           if (operation != null) {
              if (operation.equals(CREATE_CUSTOMER_ACCOUNT)) {
                 // Validate all the data introduced by the Horeca User

                 // username validation
                 if (userName != null
                       && (userName.toString().length() < USERNAME_MIN_LENGTH ||
userName
                             .toString().length()                                    >
USERNAME_PASSWORD_MAX_LENGTH)) {
                    actionErrors.add("userName", new ActionMessage(
                          INVALID_USERNAME_LENGTH));
                 }

                 // password validation
```

```
                if (password != null
                    && (password.toString().length() < PASSWORD_MIN_LENGTH ||
password
                        .toString().length()                                    >
USERNAME_PASSWORD_MAX_LENGTH)) {
                    actionErrors.add("password", new ActionMessage(
                        INVALID_PASSWORD_LENGTH));
                }

                // passwordConfirmation validation
                if (passwordConfirmation != null
                    &&          (passwordConfirmation.toString().length()          <
PASSWORD_MIN_LENGTH || passwordConfirmation
                        .toString().length()                                    >
USERNAME_PASSWORD_MAX_LENGTH)) {
                    actionErrors.add("passwordConfirmation", new ActionMessage(
                        INVALID_PASSWORD_CONFIRMATION_LENGTH));
                }

                // password and password confirmation must be the same
                if ((password != null) & (passwordConfirmation) != null
                    && (!passwordConfirmation.equals(password))) {
                    actionErrors.add("passwordConfirmation", new ActionMessage(
                        PASSWORD_ARE_NOT_THE_SAME));
                }

                // Name validation
                if (name != null
                    &&                  (name.toString().length()                  <
NAME_FIRST_SURNAME_SECOND_SURNAME_MIN_LENGTH || name
                        .toString().length()                                    >
NAME_FIRST_SURNAME_SECOND_SURNAME_MAX_LENGTH)) {
                    actionErrors.add("name", new ActionMessage(
                        INVALID_NAME_LENGTH));
                } else {
                    if (!ALPHANUMERIC_PATTERN.matcher(name).find()) {
                        actionErrors.add("name", new ActionMessage(
                            INVALID_NAME_FORMAT));
                    }
                }

                // First surname validation
                if (firstSurname != null
                    &&                (firstSurname.toString().length()            <
NAME_FIRST_SURNAME_SECOND_SURNAME_MIN_LENGTH || firstSurname
                        .toString().length()                                    >
NAME_FIRST_SURNAME_SECOND_SURNAME_MAX_LENGTH)) {
                    actionErrors.add("firstSurname", new ActionMessage(
                        INVALID_FIRST_SURNAME_LENGTH));
                } else {
                    if (!ALPHANUMERIC_PATTERN.matcher(firstSurname).find()) {
                        actionErrors.add("firstSurname", new ActionMessage(
                            INVALID_FIRST_SURNAME_FORMAT));
                    }
                }

                // Second surname validation
                if (secondSurname != null
                    &&                (secondSurname.toString().length()           <
NAME_FIRST_SURNAME_SECOND_SURNAME_MIN_LENGTH || secondSurname
                        .toString().length()                                    >
NAME_FIRST_SURNAME_SECOND_SURNAME_MAX_LENGTH)) {
                    actionErrors.add("secondSurname", new ActionMessage(
```

```
                        INVALID_SECOND_SURNAME_LENGTH));
                } else {
                    if (!ALPHANUMERIC_PATTERN.matcher(secondSurname).find()) {
                        actionErrors.add("secondSurname", new ActionMessage(
                                INVALID_SECOND_SURNAME_FORMAT));
                    }
                }

                // email validation
                if (email != null
                        && (email.toString().length() < EMAIL_MIN_LENGTH || email
                                .toString().length() > EMAIL_MAX_LENGTH)) {
                    actionErrors.add("email", new ActionMessage(
                            INVALID_EMAIL_LENGTH));
                } else {
                    Pattern p = Pattern
                            .compile("^[\\w-]+(?:\\.[\\w-]+)*@(?:[\\w-]+\\.)+[a-zA-
Z]{2,7}$");
                    Matcher m = p.matcher(email);
                    if (!m.matches()) {
                        actionErrors.add("email", new ActionMessage(
                                INVALID_EMAIL_FORMAT));
                    }
                }

                // postCode validation
                if (postCode != null
                        && (postCode.toString().length() > POST_CODE_MAX_LENGTH)) {
                    actionErrors.add("postCode", new ActionMessage(
                            INVALID_POST_CODE_LENGTH));
                } else if (!postCode.equalsIgnoreCase("")) {
                    try {
                        Long.parseLong(postCode);
                    } catch (NumberFormatException ne) {
                        actionErrors.add("postCode", new ActionMessage(
                                INVALID_POST_CODE_FORMAT));
                    }
                }
                // country, province and locality validation
                if (country.equals(-1)) {
                    actionErrors.add("country", new ActionMessage(
                            COUNTRY_NON_SELECTED));
                }

                if (province.equals(-1)) {
                    actionErrors.add("province", new ActionMessage(
                            PROVINCE_NON_SELECTED));
                }

                if (town.equals(-1)) {
                    actionErrors.add("town", new ActionMessage(
                            LOCALITY_NON_SELECTED));
                }

                // telephoneNumber validation
                if (telephoneNumber != null
                        &&              (telephoneNumber.toString().length()              >
TELEPHONE_NUMBER_MAX_LENGTH || telephoneNumber
                                .toString().length()  <  TELEPHONE_NUMBER_MIN_LENGTH))
{
                    actionErrors.add("telephoneNumber", new ActionMessage(
                            INVALID_TELEPHONE_NUMBER_LENGTH));
                } else if (!telephoneNumber.equalsIgnoreCase("")) {
```

```
                    try {
                        Long.parseLong(telephoneNumber);
                    } catch (NumberFormatException ne) {
                        actionErrors.add("telephoneNumber", new ActionMessage(
                                INVALID_TELEPHONE_NUMBER_FORMAT));
                    }
                }

                // faxNumber validation
                if (faxNumber != null
                        && faxNumber.toString().length() > FAX_NUMBER_MAX_LENGTH)
{
                    actionErrors.add("faxNumber", new ActionMessage(
                            INVALID_FAX_NUMBER_LENGTH));
                } else if (!faxNumber.equalsIgnoreCase("")) {
                    try {
                        Long.parseLong(faxNumber);
                    } catch (NumberFormatException ne) {
                        actionErrors.add("faxNumber", new ActionMessage(
                                INVALID_FAX_NUMBER_FORMAT));
                    }
                }

                // Accept usage conditons validation.
                if (acceptAllUsageConditions != true) {
                    actionErrors.add("acceptAllUsageConditions",
                            new ActionMessage(
                                    ACCEPT_USAGE_CONDITIONS_UNSELECTED));
                }
            }
        }

        WebApplicationContext ctx = WebApplicationContextUtils
                .getRequiredWebApplicationContext(request.getSession()
                        .getServletContext());

        if (!actionErrors.isEmpty()) {
            setLocations(ctx);
        }
        return actionErrors;
    }

    public void setLocations(WebApplicationContext ctx) {
        try {
            try {
                country = GeographicLocationUtils.getSpain(ctx).getId()
                        .toString();

                provinceList = GeographicLocationUtils
                        .getProvinces(GeographicLocationUtils.findLocation(ctx,
                                country));

                localityList = GeographicLocationUtils
                        .getLocalities(GeographicLocationUtils.findLocation(
                                ctx, province));
            } catch (NoSuchEntityException e) {
            }

        } catch (NullPointerException e) {
        }
    }

    /**
```

```
     * @return the userName
     */
    public String getUserName() {
        return userName;
    }

    /**
     * @param userName
     *          the userName to set
     */
    public void setUserName(String userName) {
        this.userName = userName;
    }

    /**
     * @return the password
     */
    public String getPassword() {
        return password;
    }

    /**
     * @param password
     *          the password to set
     */
    public void setPassword(String password) {
        this.password = password;
    }

    /**
     * @return the passwordConfirmation
     */
    public String getPasswordConfirmation() {
        return passwordConfirmation;
    }

    /**
     * @param passwordConfirmation
     *          the passwordConfirmation to set
     */
    public void setPasswordConfirmation(String passwordConfirmation) {
        this.passwordConfirmation = passwordConfirmation;
    }

    /**
     * @return the name
     */
    public String getName() {
        return name;
    }

    /**
     * @param name
     *          the name to set
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * @return the firstSurname
     */
    public String getFirstSurname() {
```

```
        return firstSurname;
    }

    /**
     * @param firstSurname
     *          the firstSurname to set
     */
    public void setFirstSurname(String firstSurname) {
        this.firstSurname = firstSurname;
    }

    /**
     * @return the secondSurname
     */
    public String getSecondSurname() {
        return secondSurname;
    }

    /**
     * @param secondSurname
     *          the seconacceptAllUsageConditionsdSurname to set
     */
    public void setSecondSurname(String secondSurname) {
        this.secondSurname = secondSurname;
    }

    /**
     * @return the email
     */
    public String getEmail() {
        return email;
    }

    /**
     * @param email
     *          the email to set
     */
    public void setEmail(String email) {
        this.email = email;
    }

    /**
     * @return the postCode
     */
    public String getPostCode() {
        return postCode;
    }

    /**
     * @param postCode
     *          the postCode to set
     */
    public void setPostCode(String postCode) {
        this.postCode = postCode;
    }

    /**
     * @return the countryList
     */
    public List<Location> getCountryList() {
        return countryList;
    }
```

```
/**
 * @param countryList
 *          the countryList to set
 */
public void setCountryList(List<Location> countryList) {
   this.countryList = countryList;
}

/**
 * @return the provinceList
 */
public List<Node> getProvinceList() {
   return provinceList;
}

/**
 * @param provinceList
 *          the provinceList to set
 */
public void setProvinceList(List<Node> provinceList) {
   this.provinceList = provinceList;
}

/**
 * @return the localityList
 */
public List<Node> getLocalityList() {
   return localityList;
}

/**
 * @param localityList
 *          the localityList to set
 */
public void setLocalityList(List<Node> localityList) {
   this.localityList = localityList;
}

/**
 * @return the country
 */
public String getCountry() {
   return country;
}

/**
 * @param country
 *          the country to set
 */
public void setCountry(String country) {
   this.country = country;
}

/**
 * @return the province
 */
public String getProvince() {
   return province;
}

/**
 * @param province
 *          the province to set
```

```
     */
    public void setProvince(String province) {
        this.province = province;
    }

    /**
     * @return the town
     */
    public String getTown() {
        return town;
    }

    /**
     * @param town
     *          the town to set
     */
    public void setTown(String town) {
        this.town = town;
    }

    /**
     * @return the telephoneNumber
     */
    public String getTelephoneNumber() {
        return telephoneNumber;
    }

    /**
     * @param telephoneNumber
     *          the telephoneNumber to set
     */
    public void setTelephoneNumber(String telephoneNumber) {
        this.telephoneNumber = telephoneNumber;
    }

    /**
     * @return the faxNumber
     */
    public String getFaxNumber() {
        return faxNumber;
    }

    /**
     * @param faxNumber
     *          the faxNumber to set
     */
    public void setFaxNumber(String faxNumber) {
        this.faxNumber = faxNumber;
    }

    /**
     * @return the acceptAllUsageConditions
     */
    public boolean getAcceptAllUsageConditions() {
        return acceptAllUsageConditions;
    }

    /**
     * @param acceptAllUsageConditions
     *          the acceptAllUsageConditions to set
     */
    public void setAcceptAllUsageConditions(boolean acceptAllUsageConditions) {
        this.acceptAllUsageConditions = acceptAllUsageConditions;
```

```
    }
}
```

4. Test Files (Unitary Test)

4.1. CustomerAccountDaoJpaImplTest.java (Dao Test)

```
package com.almiralabs.bookingBooks.manager.dao;

/**
 *
 */
import java.util.List;

import org.springframework.test.jpa.AbstractJpaTests;

import com.almiralabs.bookingBooks.model.CustomerAccount;
import com.almiralabs.bookingBooks.model.dao.CustomerAccountDao;
import com.almiralabs.bookingBooks.model.dao.CustomerAccountDaoJpaImpl;
import com.almiralabs.bookingBooks.model.geographic.Location;
import com.almiralabs.bookingBooks.model.nodeselection.dao.NodeDao;

/**
 * Tests for the {@link CustomerAccountDaoJpaImpl} class.
 */
public class CustomerAccountDaoJpaImplTest extends AbstractJpaTests {
    // Contants that will be used in the test.
    public static final String USERNAME001 = "user001";
    public static final String USERNAME002 = "user002";
    public static final String USERNAME003 = "user003";
    public static final String USERNAME004 = "user004";
    public static final String PASSWORD001 = "password001";
    public static final String PASSWORD002 = "password002";
    public static final String PASSWORD003 = "password003";
    public static final String PASSWORD004 = "password004";
    public static final String NAME = "Customer Name 001";
    public static final String FIRST_SURNAME = "First Surname 001";
    public static final String SECOND_SURNAME = "Second Surname 001";
    public static final String EMAIL = "customer001@almiralabs.com";
    public static final String EMAIL_2 = "customer001@gmail.com";
    public static final String COUNTRY = "España";
    public static final String PROVINCE = "Madrid";
    public static final String LOCALITY = "Torrelodones";
    public static final String TELEPHONE_NUMBER = "01234567890";

    // Daos neccesary to test this class
    public CustomerAccountDao customerAccountDao;
    public NodeDao<Location> nodeLocationDao;

    /**
     * Checks that Customer Account can be created and retrieved from database
     * once created.
```

```
         */
        public void testPersist() throws Exception {

            // Create a Location country and persist with the nodeLocationDao.
            Location country = new Location();
            country.setName(COUNTRY);
            nodeLocationDao.create(country);

            // Create Location province and persist with the nodeLocationDao.
            Location province = new Location();
            province.setName(PROVINCE);
            nodeLocationDao.create(province);

            // Create a Location locality and persist with the nodeLocationDao.
            Location locality = new Location();
            locality.setName(LOCALITY);
            nodeLocationDao.create(locality);

            // Add the province as a child of the country, and the locality as a
            // child of the province and update both objects.
            country.addChild(province);
            province.addChild(locality);
            nodeLocationDao.update(province);
            nodeLocationDao.update(locality);

            // Create a CustomeraAccount with some default values.
            CustomerAccount customerAccount001 = new
CustomerAccount(USERNAME001,
                  PASSWORD001, NAME, FIRST_SURNAME, SECOND_SURNAME, EMAIL,
                  country, province, locality, TELEPHONE_NUMBER);
            // Persist the customerAccount001.
            customerAccountDao.create(customerAccount001);

            // Retrieve from database a CustomerAccount with the same username of
            // customerAccount001.
            CustomerAccount customerAccountRetrieved = customerAccountDao
                  .loadByUsername(USERNAME001);
            // asserEquals that both CustomerAccount objects are the same.
            // Both CustomerAccount are the same, if the have the same username
            // (see the equals method of CustomerAccount Entity).
            assertEquals("Both customer Account are not the same",
                  customerAccount001, customerAccountRetrieved);
        }

        /**
         * Checks that all Customers can be retrieved from database.
         */
        public void testFindAll() throws Exception {
            // Create a Location country and persist with the nodeLocationDao
            Location country = new Location();
            country.setName(COUNTRY);
            nodeLocationDao.create(country);

            // Create a Location province and persist with the nodeLocationDao
            Location province = new Location();
            province.setName(PROVINCE);
            nodeLocationDao.create(province);

            // Create a Location locality and persist with the nodeLocationDao
            Location locality = new Location();
            locality.setName(LOCALITY);
            nodeLocationDao.create(locality);
```

```
            // Add the province as a child of the country, and the locality as a
            // child of the province and update both objects.
            country.addChild(province);
            province.addChild(locality);
            nodeLocationDao.update(province);
            nodeLocationDao.update(locality);

            // Create a CustomeraAccount customer001 with some default values.
            CustomerAccount customer001 = new CustomerAccount(USERNAME001,
                  PASSWORD001, NAME, FIRST_SURNAME, SECOND_SURNAME, EMAIL,
                  country, province, locality, TELEPHONE_NUMBER);
            customerAccountDao.create(customer001);

            // Create a CustomeraAccount customer002 with some default values.
            CustomerAccount customer002 = new CustomerAccount(USERNAME002,
                  PASSWORD002, NAME, FIRST_SURNAME, SECOND_SURNAME, EMAIL,
                  country, province, locality, TELEPHONE_NUMBER);
            customerAccountDao.create(customer002);

            // Create a CustomeraAccount customer003 with some default values.
            CustomerAccount customer003 = new CustomerAccount(USERNAME003,
                  PASSWORD003, NAME, FIRST_SURNAME, SECOND_SURNAME, EMAIL,
                  country, province, locality, TELEPHONE_NUMBER);
            customerAccountDao.create(customer003);

            // Create a CustomeraAccount customer004 with some default values.
            CustomerAccount customer004 = new CustomerAccount(USERNAME004,
                  PASSWORD004, NAME, FIRST_SURNAME, SECOND_SURNAME, EMAIL,
                  country, province, locality, TELEPHONE_NUMBER);
            customerAccountDao.create(customer004);

            // Get all the customerAccounts
            List<CustomerAccount> allCustomerAccountsFound = customerAccountDao
                  .findAll();
            // Assert as equals the number of retrieved CustomerAccounts
            assertEquals("The number of CustomerAccount retrieved is not correct",
                  4, allCustomerAccountsFound.size());
            // Assert as true that the CustomerAccount objects have been found
            assertTrue("customerAccount001 found.", allCustomerAccountsFound
                  .contains(customer001));
            assertTrue("customerAccount002 found.", allCustomerAccountsFound
                  .contains(customer002));
            assertTrue("customerAccount003 found.", allCustomerAccountsFound
                  .contains(customer003));
            assertTrue("customerAccount004 found.", allCustomerAccountsFound
                  .contains(customer004));
         }

         /**
          * Check if a CustomerAcccountEmailAddrress exist
          *
          * @throws Exception
          */
         public void testCheckCustomerAccountEmailAddressAlreadyExist() throws
Exception{
            // Create a Location country and persist with the nodeLocationDao
            Location country = new Location();
            country.setName(COUNTRY);
            nodeLocationDao.create(country);

            // Create a Location locality and persist with the nodeLocationDao
            Location province = new Location();
            province.setName(PROVINCE);
```

```
        nodeLocationDao.create(province);

        // Create a Location province and persist with the nodeLocationDao
        Location locality = new Location();
        locality.setName(LOCALITY);
        nodeLocationDao.create(locality);

        // Add the province as a child of the country, and the locality as a
        // child of the province and update both objects.
        country.addChild(province);
        province.addChild(locality);
        nodeLocationDao.update(province);
        nodeLocationDao.update(locality);

        // Create a CustomeraAccount customer001 with some default values.
        CustomerAccount customer001 = new CustomerAccount(USERNAME001,
              PASSWORD001, NAME, FIRST_SURNAME, SECOND_SURNAME, EMAIL,
              country, province, locality, TELEPHONE_NUMBER);
        customerAccountDao.create(customer001);

        // Assert as true that the email addres EMAIL already exist
        assertTrue("Customer account email address that already exist found",
              customerAccountDao
                    .checkCustomerAccountEmailAddressAlreadyExist(EMAIL));

        // Assert as false that another email addres EMAIL_2 already exist
        assertFalse("Customer account email address that already exist found",
              customerAccountDao
                    .checkCustomerAccountEmailAddressAlreadyExist(EMAIL_2));
  }
```

4.2.

CustomerAccountManagerImplTest.java

(Manager Test with JUnit)

```
/**
 *
 */
public class CustomerAccountManagerImplTest extends AbstractJpaTests {

   // Create a non-mock customerAccountDao and nodeLocationDao
   // Doing this, I am creating high-coupling with other architecture artifacts
   // I only test the method testGetAllCustomerAccounts()
   private NodeDao<Location> nodeLocationDao;

   // The CustomerAccountMager to test
   private CustomerAccountManager customerAccountManager;

   /**
    * Checks that all CustomerAccounts can be retrieved from the database.
    */
   public void testGetAllCustomerAccounts() throws Exception {

      // Create a Location country and persist with the nodeLocationDao.
      Location country = new Location();
      country.setName(CustomerAccountDaoJpaImplTest.COUNTRY);
```

```
        nodeLocationDao.create(country);

        // Create Location province and persist with the nodeLocationDao.
        Location province = new Location();
        province.setName(CustomerAccountDaoJpaImplTest.PROVINCE);
        nodeLocationDao.create(province);

        // Create a Location locality and persist with the nodeLocationDao.
        Location locality = new Location();
        locality.setName(CustomerAccountDaoJpaImplTest.LOCALITY);
        nodeLocationDao.create(locality);

        // Add the province as a child of the country, and the locality as a
        // child of the province and update both objects.
        country.addChild(province);
        province.addChild(locality);
        nodeLocationDao.update(province);
        nodeLocationDao.update(locality);

        // Create a CustomeraAccount customer001 with some default values.
        CustomerAccount customer001 = new CustomerAccount(
                CustomerAccountDaoJpaImplTest.USERNAME001,
                CustomerAccountDaoJpaImplTest.PASSWORD001,
                CustomerAccountDaoJpaImplTest.NAME,
                CustomerAccountDaoJpaImplTest.FIRST_SURNAME,
                CustomerAccountDaoJpaImplTest.SECOND_SURNAME,
                CustomerAccountDaoJpaImplTest.EMAIL, country, province,
                locality, CustomerAccountDaoJpaImplTest.TELEPHONE_NUMBER);
        customerAccountManager.create(customer001);

        // Create a CustomeraAccount customer002 with some default values.
        CustomerAccount customer002 = new CustomerAccount(
                CustomerAccountDaoJpaImplTest.USERNAME002,
                CustomerAccountDaoJpaImplTest.PASSWORD002,
                CustomerAccountDaoJpaImplTest.NAME,
                CustomerAccountDaoJpaImplTest.FIRST_SURNAME,
                CustomerAccountDaoJpaImplTest.SECOND_SURNAME,
                CustomerAccountDaoJpaImplTest.EMAIL_2, country, province,
                locality, CustomerAccountDaoJpaImplTest.TELEPHONE_NUMBER);
        customerAccountManager.create(customer002);

        // Get all the CustomerAccounts
        List<CustomerAccount> customerAccountList = customerAccountManager
                .getAllCustomerAccounts();
        // Assert as equals the number of retrieved CustomerAccounts
        assertEquals("Wrong number of Customer Accounts.", 2,
                customerAccountList.size());
        // Assert as true that the retrieved list contains customer001
        assertTrue("CustomerAccount customer001 found.", customerAccountList
                .contains(customer001));
        // Assert as true that the retrieved list contains customer001
        assertTrue("Customer Account customer002 found.", customerAccountList
                .contains(customer002));
    }
```

4.3. CustomerAccountManagerJpaImplTestMock.java (Manager Test with EasyMock)

```
/**
 *
 */
public class CustomerAccountManagerImplTestMock extends TestCase {

   private CustomerAccountDao customerAccountDaoMock;
   private CustomerAccountManagerImpl customerAccountManager;

   /**
    * {@inheritDoc}
    */
   @Override
   protected void setUp() throws Exception {
      super.setUp();
      customerAccountManager = new CustomerAccountManagerImpl();
      customerAccountManager
            .setCustomerAccountDao(customerAccountDaoMock = EasyMock
                  .createMock(CustomerAccountDao.class));
   }

   /**
    *  Test the {@link CustomerAccountManager#getAllCustomerAccounts()}
without
    * throw any Exception
    *
    * @throws Exception
    */
   public void testGetAllCustomerAccounts() throws Exception {

      List<CustomerAccount> customerAccountList = null;

      // Set the mock DAO behavior for its first call, returning a
      // List<CustomerAccount> customerAccountList.
      EasyMock.expect(customerAccountDaoMock.findAll()).andReturn(
            customerAccountList);
      EasyMock.replay(customerAccountDaoMock);

      // Manager call
      try {
         customerAccountManager.getAllCustomerAccounts();

      } catch (Exception e) {
         fail("The method testGetAllCustomerAccountsNoException can't throw any
Exception");
      }
   }

   /**
    *  Test the {@link CustomerAccountManager#getAllCustomerAccounts()}
throwing
    * the CustomerAccountManagerException
    *
    * @throws Exception
```

```
         */
        public                                                                  void
testGetAllCustomerAccountsThrowingCustomerAccountManagerException()
                throws Exception {

            // Set the mock DAO behavior for its first call, throwing a
            // DaoException()
            EasyMock.expect(customerAccountDaoMock.findAll()).andThrow(
                    new DaoException("Exception throwed in class"
                            + CustomerAccountManagerImplTestMock.class, null));
            EasyMock.replay(customerAccountDaoMock);

            // Manager call
            try {
                customerAccountManager.getAllCustomerAccounts();
                fail("The exception CustomerAccountManagerException() has not been
thrown");
            } catch (CustomerAccountManagerRuntimeException e) {
            }
        }

        /**
         * Test the {@link CustomerAccountManager#create() without throwing any
         * exception}
         *
         * @throws Exception
         */
        public void testCreate() throws Exception {

            // Create a CustomerAccount
            CustomerAccount customerAccount = new CustomerAccount(
                    CustomerAccountDaoJpaImplTest.USERNAME001,
                    CustomerAccountDaoJpaImplTest.PASSWORD001,
                    CustomerAccountDaoJpaImplTest.NAME,
                    CustomerAccountDaoJpaImplTest.FIRST_SURNAME,
                    CustomerAccountDaoJpaImplTest.SECOND_SURNAME,
                    CustomerAccountDaoJpaImplTest.EMAIL, null, null, null,
                    CustomerAccountDaoJpaImplTest.TELEPHONE_NUMBER);

            // Set the mock DAO behavior for its first call, throwing a
            // NoSuchEntityException()
            EasyMock.expect(
                    customerAccountDaoMock.loadByUsername(customerAccount
                            .getUserName())).andThrow(
                    new NoSuchEntityException(CustomerAccount.class));

            // Set the mock DAO behavior for its second call, returning false
            EasyMock
                    .expect(
                            customerAccountDaoMock

.checkCustomerAccountEmailAddressAlreadyExist(CustomerAccountDaoJpaImplTest.EMAI
L))
                    .andReturn(false);

            // Set the mock DAO behavior for its third call, returning a
            // CustomerAccount
            EasyMock.expect(customerAccountDaoMock.create(customerAccount))
                    .andReturn(customerAccount);
            EasyMock.replay(customerAccountDaoMock);
            customerAccountManager.create(customerAccount);
        }
```

```
        /**
         * Test the {@link CustomerAccountManager#create()} throwing the
         * CustomerAccountUsernameAlreadyExistException
         *
         * @throws Exception
         */
        public                                                          void
testCreateThrowingCustomerAccountUsernameAlreadyExistException()
                throws Exception {

            // Create a CustomerAccoun
            CustomerAccount customerAccount = new CustomerAccount(
                    CustomerAccountDaoJpaImplTest.USERNAME001,
                    CustomerAccountDaoJpaImplTest.PASSWORD001,
                    CustomerAccountDaoJpaImplTest.NAME,
                    CustomerAccountDaoJpaImplTest.FIRST_SURNAME,
                    CustomerAccountDaoJpaImplTest.SECOND_SURNAME,
                    CustomerAccountDaoJpaImplTest.EMAIL, null, null, null,
                    CustomerAccountDaoJpaImplTest.TELEPHONE_NUMBER);

            // Set the mock DAO behavior for its first call, returning a
            // CustomerAccount
            EasyMock.expect(
                    customerAccountDaoMock.loadByUsername(customerAccount
                            .getUserName())).andReturn(customerAccount);
            EasyMock.replay(customerAccountDaoMock);
            try {
                customerAccountManager.create(customerAccount);
                fail("Exception has been not thrown");
            } catch (CustomerAccountUsernameAlreadyExistException e) {
            }
        }

        /**
         * Test the {@link CustomerAccountManager#create()} throwing the
         * CustomerAccountEmailAddressAlreadyExistException
         *
         * @throws Exception
         */
        public                                                          void
testCreateThrowingCustomerAccountEmailAddressAlreadyExistException()
                throws Exception {

            // Create a CustomerAccount
            CustomerAccount customerAccount = new CustomerAccount(
                    CustomerAccountDaoJpaImplTest.USERNAME001,
                    CustomerAccountDaoJpaImplTest.PASSWORD001,
                    CustomerAccountDaoJpaImplTest.NAME,
                    CustomerAccountDaoJpaImplTest.FIRST_SURNAME,
                    CustomerAccountDaoJpaImplTest.SECOND_SURNAME,
                    CustomerAccountDaoJpaImplTest.EMAIL, null, null, null,
                    CustomerAccountDaoJpaImplTest.TELEPHONE_NUMBER);

            // Set the mock DAO behavior for its first call, throwing a
            // NoSuchEntityException()
            EasyMock.expect(
                    customerAccountDaoMock.loadByUsername(customerAccount
                            .getUserName())).andThrow(
                    new NoSuchEntityException(CustomerAccount.class));

            // Set the mock DAO behavior for its first call, returning true
            EasyMock
                    .expect(
```

```
                        customerAccountDaoMock

.checkCustomerAccountEmailAddressAlreadyExist(customerAccount
                                .getEmail())).andReturn(true);
            EasyMock.replay(customerAccountDaoMock);
            try {
               customerAccountManager.create(customerAccount);
               fail("Exception  CustomerAccountEmailAddressAlreadyExistException()  has
not been caught");
            } catch (CustomerAccountEmailAddressAlreadyExistException e) {
            }

         }

         /**
          * Test the {@link CustomerAccountManager#create()} throwing the
          * CustomerAccountManagerException
          *
          * @throws Exception
          */
         public void testCreateThrowingCustomerAccountManagerException()
               throws Exception {

            // Create a CustomerAccount
            CustomerAccount customerAccount = new CustomerAccount(
                  CustomerAccountDaoJpaImplTest.USERNAME001,
                  CustomerAccountDaoJpaImplTest.PASSWORD001,
                  CustomerAccountDaoJpaImplTest.NAME,
                  CustomerAccountDaoJpaImplTest.FIRST_SURNAME,
                  CustomerAccountDaoJpaImplTest.SECOND_SURNAME,
                  CustomerAccountDaoJpaImplTest.EMAIL, null, null, null,
                  CustomerAccountDaoJpaImplTest.TELEPHONE_NUMBER);

            // Set the mock DAO behavior for its first call, throwing a
            // NoSuchEntityException()
            EasyMock.expect(
                  customerAccountDaoMock.loadByUsername(customerAccount
                        .getUserName())).andThrow(
                  new NoSuchEntityException(CustomerAccount.class));

            // Set the mock DAO behavior for its first call, returning false
            EasyMock
                  .expect(
                        customerAccountDaoMock

.checkCustomerAccountEmailAddressAlreadyExist(customerAccount
                                .getEmail())).andReturn(false);
            EasyMock.expect(customerAccountDaoMock.create(customerAccount))
                  .andThrow(new InvalidEntityException(CustomerAccount.class));
            EasyMock.replay(customerAccountDaoMock);

            try {
               customerAccountManager.create(customerAccount);
               fail("Exception InvalidEntityException() has not been caught");
            } catch (CustomerAccountManagerRuntimeException e) {
            }
         }

         /**
          * Test the {@link CustomerAccountManager#create()} method throwing the
          * CustomerAccountManagerException
          *
          * @throws Exception
```

```
         */
        public void testCreateThrowingCustomerAccountManagerExceptionII()
                throws Exception {
            // Create a CustomerAccount
            CustomerAccount customerAccount = new CustomerAccount(
                    CustomerAccountDaoJpaImplTest.USERNAME001,
                    CustomerAccountDaoJpaImplTest.PASSWORD001,
                    CustomerAccountDaoJpaImplTest.NAME,
                    CustomerAccountDaoJpaImplTest.FIRST_SURNAME,
                    CustomerAccountDaoJpaImplTest.SECOND_SURNAME,
                    CustomerAccountDaoJpaImplTest.EMAIL, null, null, null,
                    CustomerAccountDaoJpaImplTest.TELEPHONE_NUMBER);

            // Set the mock DAO behavior for its first call, throwing a
            // NoSuchEntityException()
            EasyMock.expect(
                    customerAccountDaoMock.loadByUsername(customerAccount
                            .getUserName())).andThrow(
                    new NoSuchEntityException(CustomerAccount.class));

            // Set the mock DAO behavior for its first call, returning false
            EasyMock
                    .expect(
                            customerAccountDaoMock

.checkCustomerAccountEmailAddressAlreadyExist(customerAccount
                                    .getEmail())).andReturn(false);

            // Set the mock DAO behavior for its first call, throwing a
            // DuplicatedEntityException()
            EasyMock.expect(customerAccountDaoMock.create(customerAccount))
                    .andThrow(new DuplicatedEntityException(CustomerAccount.class));

            EasyMock.replay(customerAccountDaoMock);

            try {
                customerAccountManager.create(customerAccount);
                fail("Exception DuplicatedEntityException() has not been caught");
            } catch (CustomerAccountManagerRuntimeException e) {
            }
        }

        /**
         * Test the {@link CustomerAccountManager#loadByUsername()} method
         *
         * @throws Exception
         */
        public void testLoadByUsername() throws Exception {

            // Create a CustomerAccount
            CustomerAccount customerAccount = new CustomerAccount(
                    CustomerAccountDaoJpaImplTest.USERNAME001,
                    CustomerAccountDaoJpaImplTest.PASSWORD001,
                    CustomerAccountDaoJpaImplTest.NAME,
                    CustomerAccountDaoJpaImplTest.FIRST_SURNAME,
                    CustomerAccountDaoJpaImplTest.SECOND_SURNAME,
                    CustomerAccountDaoJpaImplTest.EMAIL, null, null, null,
                    CustomerAccountDaoJpaImplTest.TELEPHONE_NUMBER);

            // Set the mock DAO behavior for its first call, returning a
            // customerAccount
            EasyMock
                    .expect(
```

```
                        customerAccountDaoMock

.loadByUsername(CustomerAccountDaoJpaImplTest.USERNAME001))
                .andReturn(customerAccount);
            EasyMock.replay(customerAccountDaoMock);
            customerAccountManager
                .loadByUsername(CustomerAccountDaoJpaImplTest.USERNAME001);
        }

        /**
         * Test the {@link CustomerAccountManager#loadByUsername()} throwing
         * CustomerAccountManagerException
         *
         * @throws Exception
         */
        public void testLoadByUsernameThrowingCustomerAccountManagerExceptionI()
                throws Exception {
            // Set the mock DAO behavior for its first call, throwing a
            // DaoException()
            EasyMock
                .expect(
                    customerAccountDaoMock

.loadByUsername(CustomerAccountDaoJpaImplTest.USERNAME001))
                .andThrow(new DaoException(null));
            EasyMock.replay(customerAccountDaoMock);

            try {
                customerAccountManager
                    .loadByUsername(CustomerAccountDaoJpaImplTest.USERNAME001);
                fail("Exception DuplicatedEntityException() has not been caught");
            } catch (CustomerAccountManagerRuntimeException e) {
            }
        }

        /**
         * Test the {@link CustomerAccountManager#loadByUsername()} throwing
         * CustomerAccountManagerException
         *
         * @throws Exception
         */
        public void testLoadByUsernameThrowingCustomerAccountManagerExceptionII()
                throws Exception {
            // Set the mock DAO behavior for its first call, throwing a
            // NoSuchEntityException()
            EasyMock
                .expect(
                    customerAccountDaoMock

.loadByUsername(CustomerAccountDaoJpaImplTest.USERNAME001))
                .andThrow(new NoSuchEntityException(null));
            EasyMock.replay(customerAccountDaoMock);
            try {
                customerAccountManager
                    .loadByUsername(CustomerAccountDaoJpaImplTest.USERNAME001);
                fail("Exception DuplicatedEntityException() has not been caught");
            } catch (CustomerAccountManagerRuntimeException e) {
            }
        }
    }
```

4.4. ManageCustomersActionTest.java (Struts Action Test)

```
/**
 * Tests for the {@link ManageCustomersAction} Struts action.
 *
 * @author Ginés Sánchez
 *
 */

public class ManageCustomersActionTest extends MockStrutsTestCase {

   private TestWebApplicationContext wac = null;
   private static final String OPERATION = "operation";
   public static final String USERNAME001 = "user001";
   public static final String USERNAME002 = "user002";
   public static final String USERNAME003 = "user003";
   public static final String USERNAME004 = "user004";
   public static final String PASSWORD001 = "password001";
   public static final String PASSWORD002 = "password002";
   public static final String PASSWORD003 = "password003";
   public static final String PASSWORD004 = "password004";
   public static final String NAME = "Customer Name 001";
   public static final String FIRST_SURNAME = "First Surname 001";
   public static final String SECOND_SURNAME = "Second Surname 001";
   public static final String EMAIL = "customer001@almiralabs.com";
   public static final String EMAIL_2 = "customer001@gmail.com";
   public static final String POST_CODE = "28290";
   public static final String COUNTRY = "España";
   public static final String PROVINCE = "Madrid";
   public static final String LOCALITY = "Torrelodones";
   public static final String TELEPHONE_NUMBER = "01234567890";
   public static final String FAX_NUMBER = "09876543210";

   /**
    * Tests the
    *                                                                    {@link
ManageCustomersAction#create(org.apache.struts.action.ActionMapping,
org.apache.struts.action.ActionForm,          javax.servlet.http.HttpServletRequest,
javax.servlet.http.HttpServletResponse)}
    * without throw any exceptions
    *
    * @throws ExceptionException
    *           If something goes wrong.
    */
   public void testCreate() throws Exception {
      // Create the CustomerAccountManager
      CustomerAccountManager customerAccountManagerMock = EasyMock
            .createMock(CustomerAccountManager.class);
      // Create the ActionForm and set its values
      ManageCustomersActionForm      actionForm      =      new
ManageCustomersActionForm();
      actionForm.setUserName(USERNAME001);
      actionForm.setPassword(PASSWORD001);
      actionForm.setPasswordConfirmation(PASSWORD001);
      actionForm.setName(NAME);
      actionForm.setFirstSurname(FIRST_SURNAME);
      actionForm.setSecondSurname(SECOND_SURNAME);
      actionForm.setCountry(COUNTRY);
```

```
            actionForm.setProvince(PROVINCE);
            actionForm.setTown(LOCALITY);
            actionForm.setPostCode(POST_CODE);
            actionForm.setEmail(EMAIL);
            actionForm.setTelephoneNumber(TELEPHONE_NUMBER);
            actionForm.setFaxNumber(FAX_NUMBER);
            actionForm.setAcceptAllUsageConditions(true);

            // Create the CustomerAccount
            CustomerAccount customerAccount = new CustomerAccount(actionForm
                  .getUserName(), actionForm.getPassword(), actionForm.getName(),
                  actionForm.getFirstSurname(), actionForm.getSecondSurname(),
                  actionForm.getEmail(), new Location(), new Location(),
                  new Location(), actionForm.getTelephoneNumber());

            // Set the Manager behavior for its first call, returning a
            // customerAccount
            EasyMock.expect(customerAccountManagerMock.create(customerAccount))
                  .andReturn(customerAccount);
            EasyMock.replay(customerAccountManagerMock);

            // Register the beans
            registerBean("customerAccountManager", customerAccountManagerMock);
            // Set request parameters
            setRequestPathInfo("/createNewCustomerAccount");
            addRequestParameter(OPERATION, "create");
            // Set actionForm
            setActionForm(actionForm);
            actionPerform();
            verifyForward(ManageCustomersAction.MAPPING_CREATE_SUCCESS);
            verifyActionMessages(new                    String[]                    {
ManageCustomersAction.CUSTOMER_ACCOUNT_SUCCESSFULLY_CREATED });
         }

         /**
          * Tests the
          *                                                                {@link
ManageCustomersAction#create(org.apache.struts.action.ActionMapping,
org.apache.struts.action.ActionForm,          javax.servlet.http.HttpServletRequest,
javax.servlet.http.HttpServletResponse)}
          *   throw   the   exception   CustomerAccountUsernameAlreadyExistException
because
          * the CustomerAccount username already exist
          *
          * @throws Exception
          *           If something goes wrong.
          */
         public void testCreateThrowCustomerAccountUsernameAlreadyExistException()
               throws Exception {
            // Create the CustomerAccountManager
            CustomerAccountManager customerAccountManagerMock = EasyMock
                  .createMock(CustomerAccountManager.class);
            // Create the ActionForm and set its values
            ManageCustomersActionForm          actionForm          =          new
ManageCustomersActionForm();
            actionForm.setUserName(USERNAME001);
            actionForm.setPassword(PASSWORD001);
            actionForm.setPasswordConfirmation(PASSWORD001);
            actionForm.setName(NAME);
            actionForm.setFirstSurname(FIRST_SURNAME);
            actionForm.setSecondSurname(SECOND_SURNAME);
            actionForm.setCountry(COUNTRY);
            actionForm.setProvince(PROVINCE);
```

```
            actionForm.setTown(LOCALITY);
            actionForm.setPostCode(POST_CODE);
            actionForm.setEmail(EMAIL);
            actionForm.setTelephoneNumber(TELEPHONE_NUMBER);
            actionForm.setFaxNumber(FAX_NUMBER);
            actionForm.setAcceptAllUsageConditions(true);
            actionForm.setLocations(null);

            // Create the CustomerAccount
            CustomerAccount customerAccount = new CustomerAccount(actionForm
                  .getUserName(), actionForm.getPassword(), actionForm.getName(),
                  actionForm.getFirstSurname(), actionForm.getSecondSurname(),
                  actionForm.getEmail(), new Location(), new Location(),
                  new Location(), actionForm.getTelephoneNumber());

            // Set the Manager behavior for its first call, throwing a
            // CustomerAccountUsernameAlreadyExistException()
            EasyMock.expect(customerAccountManagerMock.create(customerAccount))
                  .andThrow(
                        new CustomerAccountUsernameAlreadyExistException(
                              "Customer Account Username Already Exist"));
            EasyMock.replay(customerAccountManagerMock);

            // Register the beans
            registerBean("customerAccountManager", customerAccountManagerMock);
            // Set request parameters
            setRequestPathInfo("/createNewCustomerAccount");
            addRequestParameter(OPERATION, "create");
            setActionForm(actionForm);
            // Set actionForm
            actionPerform();

verifyForward(ManageCustomersAction.FORWARD_USERNAME_ALREADY_TAKEN);
            verifyActionErrors(new                    String[]                    {
ManageCustomersAction.ERROR_USERNAME_ALREADY_TAKEN });
      }

      /**
       * Tests the
       *                                                                  {@link
ManageCustomersAction#create(org.apache.struts.action.ActionMapping,
org.apache.struts.action.ActionForm,          javax.servlet.http.HttpServletRequest,
javax.servlet.http.HttpServletResponse)}
       * throw the exception CustomerAccountEmailAddressAlreadyExistException
       * because the CustomerAccount email address already exist
       *
       * @throws Exception
       *           If something goes wrong.
       */
      public                                                                  void
testCreateThrowCustomerAccountEmailAddressAlreadyExistException()
            throws Exception {
            // Create the CustomerAccountManager
            CustomerAccountManager customerAccountManagerMock = EasyMock
                  .createMock(CustomerAccountManager.class);
            // Create the ActionForm and set its values
            ManageCustomersActionForm          actionForm          =          new
ManageCustomersActionForm();
            actionForm.setUserName(USERNAME001);
            actionForm.setPassword(PASSWORD001);
            actionForm.setPasswordConfirmation(PASSWORD001);
            actionForm.setName(NAME);
            actionForm.setFirstSurname(FIRST_SURNAME);
```

```
            actionForm.setSecondSurname(SECOND_SURNAME);
            actionForm.setCountry(COUNTRY);
            actionForm.setProvince(PROVINCE);
            actionForm.setTown(LOCALITY);
            actionForm.setPostCode(POST_CODE);
            actionForm.setEmail(EMAIL);
            actionForm.setTelephoneNumber(TELEPHONE_NUMBER);
            actionForm.setFaxNumber(FAX_NUMBER);
            actionForm.setAcceptAllUsageConditions(true);

            // Create the CustomerAccount
            CustomerAccount customerAccount = new CustomerAccount(actionForm
                  .getUserName(), actionForm.getPassword(), actionForm.getName(),
                  actionForm.getFirstSurname(), actionForm.getSecondSurname(),
                  actionForm.getEmail(), new Location(), new Location(),
                  new Location(), actionForm.getTelephoneNumber());
            // Set the Manager behavior for its first call, throwing a
            // CustomerAccountEmailAddressAlreadyExistException()
            EasyMock
                  .expect(customerAccountManagerMock.create(customerAccount))
                  .andThrow(
                        new CustomerAccountEmailAddressAlreadyExistException(
                              "Customer Account Email Address Already Taken"));
            EasyMock.replay(customerAccountManagerMock);
            // Register the beans
            registerBean("customerAccountManager", customerAccountManagerMock);
            setRequestPathInfo("/createNewCustomerAccount");
            addRequestParameter(OPERATION, "create");
            // Set actionForm
            setActionForm(actionForm);
            actionPerform();
            verifyForward(ManageCustomersAction.FORWARD_EMAIL_ALREADY_TAKEN);
            verifyActionErrors(new                    String[]                    {
ManageCustomersAction.ERROR_EMAIL_ALREADY_TAKEN });
      }

      @Override
      protected void setUp() throws Exception {
         super.setUp();
         // bind the WebApplicationContext to the servletContext
         ServletContext sc = getActionServlet().getServletContext();
         wac = new TestWebApplicationContext();
         wac.setServletContext(sc);
         wac.refresh();
         sc.setAttribute(

StaticWebApplicationContext.ROOT_WEB_APPLICATION_CONTEXT_ATTRIBUTE,
                  this.wac);
      }

      protected void registerBean(String key, Object value) {
         wac.addBean(key, value);
      } }
```

5. Configuration Files

5.1. Spring-bookingBoks-config.xml (Spring)

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:tx="http://www.springframework.org/schema/tx"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
                  http://www.springframework.org/schema/beans/spring-beans-
2.5.xsd"
   default-autowire="no">

   <!-- DAOs -->
   <bean id="customerAccountDao"

class="com.almiralabs.bookingbooks.model.dao.CustomerAccountDaoJpaImpl"
      lazy-init="true" />

   <!-- Managers -->
   <bean id="customerAccountManager"

class="com.almiralabs.bookingbooks.manager.CustomerAccountManagerImpl"
      scope="prototype">
      <property name="customerAccountDao">
         <ref local="customerAccountDao" />
      </property>
   </bean>

</beans>
```

5.2. Struts-config.xml (Struts)

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC
      "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
      "http://struts.apache.org/dtds/struts-config_1_2.dtd">
<struts-config>

   <form-beans>
      <!-- CONCEPT -->
      <form-bean name="manageCustomersActionForm"

type="com.almiralabs.bookingbooks.ui.controller.ManageCustomersActionForm" />
      <form-bean name="loginCustomerActionForm"

type="com.almiralabs.bookingbooks.ui.controller.LoginCustomerAccountActionForm" />
      <form-bean name="createPdfFilesActionForm"

type="com.almiralabs.bookingbooks.ui.controller.CreatePdfFilesActionForm" />
   </form-beans>

  <action-mappings>
```

```
            <action path="/home"
               forward="forward.tiles.index" />

            <action path="/customer"
               forward="forward.tiles.create.new.customer.account" />

            <action path="/products"
               forward="forward.tiles.products" />

            <action path="/showProtectionDataClause"
               forward="forward.tiles.protection.data.clause" />

            <action path="/showUsageConditionsTerms"
               forward="forward.tiles.tems.and.conditions" />

            <action path="/about"
               forward="forward.tiles.about" />

            <!--
============================================================
====== -->

            <!-- Booking BOOKS shared Actions -->
            <action path="/obtainGeographicData"
               type="com.almiralabs.bookingbooks.ui.controller.GeographicXmlAction" />

            <!-- Customer Main -->
            <action path="/customermain"
               forward="forward.tiles.index" />

            <action path="/createNewCustomerAccount"
               type="com.almiralabs.bookingbooks.ui.controller.ManageCustomersAction"
               parameter="operation"
               name="manageCustomersActionForm"

input="/createNewCustomerAccount.do?operation=showFormNewCustomerAccount"
                  scope="request" validate="true">

               <forward name="preapareNewAccount"
                  path="forward.tiles.create.new.customer.account" />
               <forward name="success"
                  path="forward.tiles.index" />
               <forward name="usernameAlreadyTaken"
                  path="forward.tiles.create.new.customer.account" />
               <forward name="emailAlreadyTaken"
                  path="forward.tiles.create.new.customer.account" />
            </action>

            <action path="/createPdfFiles"
               type="com.almiralabs.bookingbooks.ui.controller.CreatePdfFilesAction"
               parameter="operation"
               name="createPdfFilesActionForm"
               input="/createNewCustomerAccount.do"
                  scope="request" validate="true">
               <forward name="successProtectionDataClause"
                  path="/WEB-INF/views/bookingbooks/ProtectionDataClausePDF.jsp" />
               <forward name="successTermsAndConditions"
                  path="/WEB-INF/views/bookingbooks/TermsAndConditionsPDF.jsp" />
            </action>

            <action path="/loginCustomerAccount"

type="com.almiralabs.bookingbooks.ui.controller.LoginCustomerAccountAction"
```

```
                parameter="operation"
                name="loginCustomerActionForm"
                input="/customermain.do"
                   scope="request" validate="true">

                <forward name="successLogin"
                   path="forward.tiles.index" />
                <forward name="errorLogin"
                   path="forward.tiles.index" />
             </action>

          </action-mappings>
          <message-resources
             parameter="com.almiralabs.bookingBooks.ui.user-interface-messages"
             null="false" />

          <!--Tiles plugin-->
          <plug-in className="org.apache.struts.tiles.TilesPlugin">
             <set-property  property="definitions-config"  value="/WEB-INF/tiles-defs.xml"
/>
          </plug-in>
       </struts-config>
```

ANNEX II

ANNEX II: Project Execution

1. Project Budget

1.1. General Budget

1. Autor	Ginés Sánchez Bustillo
2. Departamento	Desarrollo de Aplicaciones Informáticas
3. Descripción del proyecto	**Título:** GUIDELINE TO APPLY THE ISO 90003:2004 STANDARD TO SMEs OF SOFTWARE DEVELOPMENT
	Duración: 602 horas; es decir 16 semanas; o bien 3 meses y 3 semanas.
4. Presupuesto total del proyecto	No presupuestado.
5. A) Desglose presupuesto (costes directos)	**Apellidos y nombre:** Ginés Sánchez Bustillo.
	NIF: no es necesario.
	Categoría: Ingeniero Técnico Informático Informático, Desarrollador Junior.
	Dedicación: completa, 160 horas/mes.
	Coste hombre/mes: 2500 €.
	Firma de conformidad:
	TOTAL COSTE: 2500 €/mes * 3 meses y 3 semanas: 9.375,00 €
5. B) Equipos	**Descripción:** Ordenador portátil Hacer Aspire 5104 WLMi.
	Coste: 700 €.
	% Uso dedicado: 100%.
	Dedicación: 3 meses y 3 semanas.
	Periodo de depreciación: 60
	TOTAL COSTE: 35,20 €

5. C) Subcontratación de tareas	**Descripción:** -
	Empresa: -
	TOTAL COSTE: 0,00 €.
5. D) Costes de funcionamiento	Desplazamientos, licencias software propietario, adquisición de normas, agua, electricidad.
	TOTAL COSTE: 800,00 €
6. Resumen de los costes	Personal: 9.375,00 €
	Amortización: 35,20 €
	Subcontratación de tareas: 0,00 €
	Costes de funcionamiento: 800,00 €
	Costes indirectos: 2.042,04 €
	TOTAL COSTE: 12.252,24 €

1.2. Issues Description

Issue Name	Issue Start Date	Duration (hours)
Start of the first version of the project: AUDITORÍA Y CONTROL DE UN ENTORNO DE DESARROLLO DE APLICACIONES SOFTWARE SEGÚN LA NORMA ISO 90003:2004 (spanish version).	15/09/2007	80
First meeting with Antonio García Carmona.	8/10/2008	5
Start of the second version of the project: Guideline to apply the ISO 90003:2004 to SMEs of software development.	18/10/2008	12
Change of language from Spanish to English.	18/10/2208	10
Creation of Xuse XML files with application description, installation, coding standards, release process...	21/10/2008	60
User stories creation	12/11/2008	8
Requirement elicitation process	26/11/2008	10
Creation of the Web Application code archetype	01/09/2009	40
Use Case 001: Customer Registration Specification and Implementation	11/09/2009	48
Use Case 002: Customer Password Recovery Specification	25/09/2009	10

Use Case 003: Customer Account Specification	26/09/2009	10
Use Case 004: Customer Books Search Specification	27/09/2009	10
Use Case 005: Customer Login Specification	28/09/2009	10
Creation and configuration of the css files for the view	29/09/2009	30
Second meeting with Antonio García Carmona.	28/10/2009	2
Fix of some some errors in core part and test classes.	1/11/2009	8
Review of the application behavior; test cycle.	8/11/2009	18
End of the Web Application Development.	12/11/2009	
Architecture Definition.	5/06/2010	14
Tools Definition.	8/06/2010	25
Key Concepts defininition.	16/06/2010	6
Deleted CMMi as maturity model. Created document definition template. Changed ISO 9001:2000 for 9001:2008, added index of pictures. Added part "Application guideline tools".	9/09/2010	100
Review of the project. Deleted some "It is not necessary add any guideline". Added some application tools.	7/10/2010	40
Added numeration, project reviewed again.	20/10/2010	14
Final review, added budget estimation	5/11/2010	12
Final meeting with Antonio García Carmona.	22/10/2010	20
TOTAL		602

2. Meeting Minutes

2.1. First Meeting Minutes

ACTA REUNIÓN

Número de la sesión: 1ª.	**Fecha:** Miércoles, 8 De Octubre de 2008.
Lugar: Despacho Departamento Informática. Edificio Miguel de Unamuno. Campus De Colmenarejo. Universidad Carlos III. Madrid.	**Horario**: 16:20 a 16:50 Horas.
Propósito: Primera reunión formal entre las partes. Discusión, definición y estudio del Proyecto Fin de Carrera a partir de un documento de aproximación.	

Asistentes:
Tutor: **D. Antonio García Carmona**
Desarrollador: **D. Ginés Sánchez Bustillo**

Orden del día:

1. Revisión del documento de aproximación al Proyecto Fin de Carrera desarrollado como fase introductoria.
2. Resolución de dudas respecto a:
   - Título del proyecto.
   - Discusión de la estructura básica a partir del índice, así como de cada uno de los apartados.
   - Requisitos que deben ser cumplidos en su desarrollo, objetivos fundamentales a cubrir, planificación aproximada.
   - Referencias y comentarios a otras normas de calidad / estándares.
   - Entorno de aplicaciones de desarrollo en el que proyecto estará enmarcado.
   - Tipo de organización al que irá dirigido.
   - Idioma en el que será realizado.

Desarrollo de la sesión:

1. Como primera revisión del trabajo realizado hasta ahora en el documento de aproximación, se trata de revisar y verificar si lo que se ha realizado es correcto y coherente con los requisitos del Proyecto.
2. Se realiza un repaso al mismo, fundamentalmente al índice, extrayendo una serie de requisitos y defectos de este documento inicial.
3. Rápidamente surgen carencias que se matizan en el siguiente detalle:
   - . El título del Proyecto ha de cambiar de "AUDITORÍA Y CONTROL DE UN ENTORNO DE DESARROLLO DE APLICACIONES SOFTWARE SEGÚN LA NORMA ISO 9001:2000" a "ADAPTACIÓN DE UN ENTORNO DE DESARROLLO DE APLICACIONES WEB SEGÚN LA NORMA ISO 90003:2004".
   - La finalidad del proyecto consiste en que una pequeña/mediana empresa (PYME) de desarrollo de software Web pueda obtener la certificación de calidad ISO 9001:2000 (la 90003:2004 no es certificable como tal), teniendo en cuenta los limitados recursos de los que, a priori, pueda disponer.
   - El proyecto estará enmarcado hacia un tipo desarrollo de aplicaciones Web, tomando como principio la tecnología Java (J2EE), ya que es la predominante en este tipo de entornos. Aún y así, se hace constar que no se

hará dependiente en ningún caso de esta tecnología, por lo que será aplicable a otros lenguajes / entornos de desarrollo, siempre teniendo en cuenta la necesidad de ser un lenguaje conceptualmente compatible (.NET, PHP, etc).

• Se tendrá en cuenta los limitados recursos de los que dispone una PYME, de manera que se hará especial hincapié en la utilización de software de bajo coste de propiedad, tanto en el sistema operativo (Ubuntu GNU-Linux) como herramientas y aplicaciones, dando especial importancia a productos bajo la licencia GNU GPL y proyectos similares.

• A petición del alumno, el proyecto será desarrollado íntegramente en Inglés, (conteniendo un resumen del mismo de unas diez páginas en español), por ser el idioma predominante en estos desarrollos informáticos, si bien la defensa se realizará en español.

• El Índice del Proyecto será prácticamente idéntico a la norma ISO 90003, con otros puntos adicionales y complementarios al proyecto, y contenidos con referencia a ella. Se acuerda hacerlo así para cubrir de manera amplia y concreta la citada norma, aunque se podrán hacer referencias a otras.

• En el Proyecto se incluirá, un caso práctico de desarrollo; desde un requisito inicial (o hasta cinco requisitos) expuestos por el cliente hasta su "desplegable" o "ejecutable"; lo importante será aquí cómo crear y configurar las herramientas básicas e infraestructura tecnológica que de soporte a ese desarrollo.

• No se incluirán en el desarrollo del Proyecto "hojas check" a modo de auditoria, ya que se entiende que si la organización acepta e implementa las herramientas propuestas el fin último de garantizar la calidad (y por tanto certificarse) será cumplido.

• Se adjuntará con el proyecto documentación asociada a su desarrollo, como actas de reunión, planificación básica con los distintos hitos y entregables, y documentos de control de cambios.

Objetivos a alcanzar en próximas reuniones:

1. Revisión del desarrollo de la versión 1.0.
2. Fecha planificada: Enero-Febrero 2009.

Se levanta la sesión, de la cual se extiende a todas las partes por correo electrónico el presente acta, con el mutuo acuerdo de:

DESARROLLADOR	**Tutor**
D. Ginés Sánchez Bustillo	D. Antonio García Carmona

2.2. Second Meeting Minutes

ACTA REUNIÓN

Número de la sesión: 2ª.	**Fecha:** Miércoles, 28 De Octubre de 2009.
Lugar: Aula 11A16. Edificio Miguel de Unamuno. Campus De Colmenarejo. Universidad Carlos III. Madrid.	**Horario**: 16:00 a 16:50 Horas.
Propósito: Segunda reunión formal entre las partes. Revisión de la Aplicación Web, así como de el documento del proyecto final de carrera.	

Asistentes:
Tutor: **D. Antonio García Carmona**
Desarrollador: **D. Ginés Sánchez Bustillo**

Orden del día:

1. Revisión de la Aplicación Web, así como de los requisitos, casos de uso, historias de usuario; funcionamiento global de la misma .
2. Resolución de dudas respecto a:
   - Formato del proyecto fin de carrera
   - Planificación del mismo.

Desarrollo de la sesión:

1. Como segunda revisión el alumno enseña la Aplicación Web al tutor; se realiza una navegación por la misma y se muestran aspectos arquitectónicos, pruebas unitarias y de integración, así como toda la documentación generada con Xuse (Casos de uso, requisitos, etc).
2. A pesar de algunos fallos en las pruebas unitarias, el tutor muestra su satisfacción por el desarrollo global de la aplicación, dando algunas directrices generales acerca de la memoria del mismo, que deberán ser aplicadas posteriormente.

Objetivos a alcanzar en próximas reuniones:

1. Ejecución de la aplicación, así como de sus test unitarios sin ningún tipo de errores.
2. Revisión de el documento del proyecto fin de carrera con todas las secciones correctamente completadas.
3. Fecha planificada: Marzo-Abril 2010.

Se levanta la sesión, de la cual se extiende a todas las partes por correo electrónico el presente acta, con el mutuo acuerdo de:

DESARROLLADORES	Tutor
D. Ginés Sánchez Bustillo	D. Antonio García Carmona