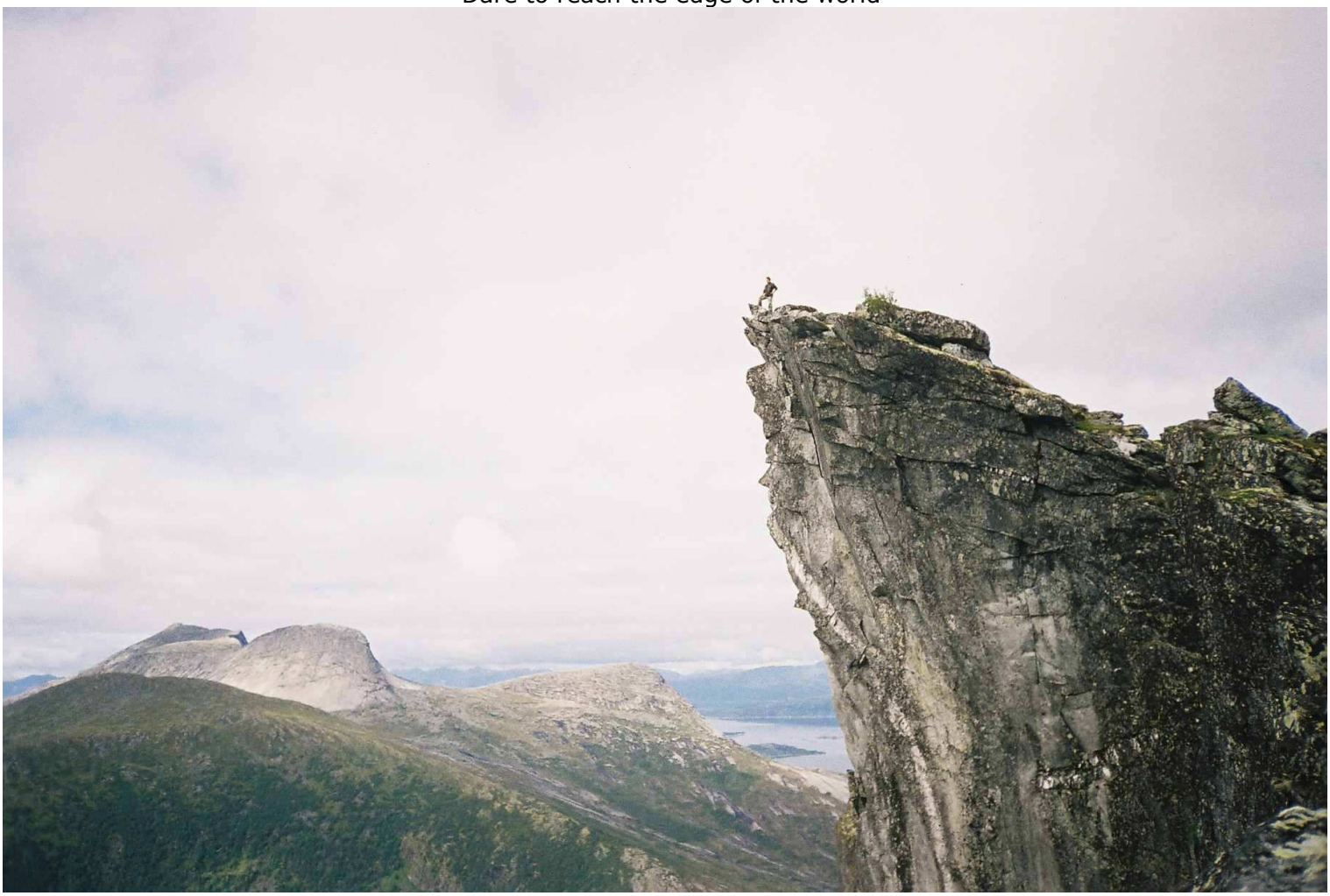
Study Project Report
For developing a web-oriented solution for Roskilde Grej

Group: 3
Members: Vilma Rudžionytė, Darius Damalakas
Date: 05 12 2004

Roskilde Business College, 2004
3rd semester, Advanced Computer Science

Dare to reach the edge of the world



Kauskas round the Baltic
Norway, 2004 summer, ©

1 Introduction

This is the Study Project Report for team 2's 3rd semester system development and programming project.

Team 2 consists of the following 2 members: Darius and Vilma.

We are working together for a common goal; to learn as much as possible during one semester.

Our project this semester is to provide a web-oriented (also termed e-solution) for the company Roskilde Grej. This provides us with a unique and challenging experience of working on a real world problem within the structure of our education.

The following report documents the development of our project including the decisions that we have made, the challenges that we have faced, and ultimately what we have learned.

Table of contents

1Introduction....3
2Project Start....6
3Problem Analysis....7
3.1Problem Definition....7
4Project Scope....8
4.1Project requirements....8
4.2Project focus....8
5Project start....9
5.1Document outline....9
5.2Project methodology....9
6User participation and user interface design....11
6.1Importance of user participation....11
6.2Level of user participation....11
6.3User participation problems....12
6.4Ways of improving user participation and involvement....12
6.5User participation in Rewebution project....13
6.6Design Brief....13
6.6.1Why provide a design brief? 13
6.6.2Design Brief instances....13
6.7Navigation map....14
6.7.1Why need Navigation map?....14
6.7.2Navigation map in our project....14
6.8Design Comps 15
6.8.1Purpose of design comps....15
6.8.2 Design comps instances....15
6.9User interface evaluation....16
6.10Web Design Elements....17
6.10.1The purpose of web design elements....17
6.10.2Web Design Elements in our project....17
6.11User interface description....17
7Client-Server architecture 19
7.1Introduction to client-server architecture....19
7.2Definition of the Web Application....19
7.3Three common used web-application architectural patterns....19
7.3.1 Core components....20
7.3.2 Common additional components....20
7.4Thin Web Client....20
7.4.1 Application domain 20
7.4.2 Logic....20
7.4.3 Logical view....21
7.5Thick Web Client....22
7.5.1Application Domain....22
7.5.2Logic....22
7.5.3Logical View....22
7.6Web Delivery....23
7.6.1Application Domain....23
7.6.2Logic....23
7.6.3Logical View....23
7.7Overview of the three architectural patterns....24
7.8Content management systems....25
7.9KillBenas web application with CMS....25
7.10Architectural pattern of Rewebution System....27
7.11Designing the system....27
7.11.1 UC102 View Equipment List Home....27

7.11.2 UC107 Put Equipment To Cart.....29
8Web programming 31
8.1Introduction to web programming.....31
8.2HTML.....31
8.3Java component models 31
8.4Accessible objects to JSP and servlets.....32
8.5Java servlet defined.....32
8.6JavaBeans and Enterprise JavaBeans.....35
8.7JSP defined.....35
8.8Model View Controller architecture.....36
8.9 Implement MVC architecture with forwarding or redirecting.....37
8.10 Java scriptlets and java expressions.....37
8.11 JSP expression language (EL) elements.....38
8.12 JSP tags.....38
8.13JavaBeans.....40
8.14RMI.....42
8.15Sessions.....44
8.16The HttpSession and ServletContext objects application.....45
8.17Database connections and connection pooling.....45
8.17.1Hand made approach.....46
8.17.2Using JNDI and DataSource.....46
8.17.3Rewebution approach.....47
9Conclusion.....48
10Literature.....49
11Appendices.....50
11.1Appendix A - "XP versus UP – a methodology comparison study work".....50

2 Project Start

The Rewebution[1] will be the system we are going to investigate and build for Roskilde Grej. The system will be built using the database, which was developed during the 2nd semester project for the "Rental System". Rewebution will be a web project – a system for allowing rental to be made on-line.

Rewebution will communicate with the Rental System and be implemented with client-server architecture. As we are who did the Rental System previous semester, it will be easier for us to investigate and adapt the Rental System so it will be able to communicate with Rewebution system easily. Significant amount of analysis work was already done, so we will reuse as much as possible information from Rental System project.

[1] Rewebution – REservation system WEB-solUTION.

3 Problem Analysis

3.1 Problem Definition

The purpose of our report is while using the external project requirements given to us by Roskilde Grej, we are to answer the problem question, which is "How can the methodology "WEB-RUP" and web-programming support the work of developing a web-based multi user system for a company".

4 Project Scope

4.1 Project requirements

It is required that we build the Rewebution system by use of Unified Process as it is described in "Building Web Solutions with the Rational Unified Process: Unifying the Creative Design Process and the Software Engineering Process" , a Rational Software and Context Integration white paper and selected pages from "Building WEB applications with UML" by Jim Conallen.

It is required that the design is realized by means of the programming languages HTML and JSP. IT is required that a relational database is used in the system.

4.2 Project focus

By the means of building a Rental System for Roskilde Grej Company in 2nd semester and problem definition, we state that we will focus on the following topics in this project:

- User participation
- User interface design
- Client-Server architecture
- Web programming

As stated in project requirements, we are to use a "Web-RUP" methodology. This conveys a hand-full of deliverables will be produced. However, the project will discuss and explain only those documents and system parts which are relevant for this project because of project report size limit (total of 60 pages). Other deliverables will be excluded, though we will follow the process as it is stipulated in corresponding documents. The excluded documents will be primarily from management discipline (Risk List, Software development Plan and corresponding) and some from requirements (the use case model).
We will focus on understanding the overall overview of a handful of technologies for the topics we have chosen. We will try also to highlight the essential differences between the technologies addressing the same domain area, thus we will not discuss details as much thoroughly as we could (this decision is also exposed by the limit of the project report size).

5 Project start

5.1 Document outline

Since our goal is more to understand the technologies and discuss the possible uses in our project, we tried to structure the document so as the theory would be the main focus and would come before the application. Carrying out our project management activities we have planned the project as follows:

- The inception phase would be more about understanding the requirements for the Roskilde Grej and designing the user interface. Hence the following topics are the area of interest - User participation and User interface design.
- The elaboration phase first iteration will be primarily devoted to finishing up the user interface design and defining the architecture of the forthcoming system. Thus the Client-Server architecture is the area of interest.
- In the last iteration of elaboration phase we are to implement some of the use cases based on our user interface design and the architecture. The system will be operational on the net and thus Web programming is the topic to be discussed thoroughly.

5.2 Project methodology

We are to use in our project "Web-RUP" methodology as defined by project requirements. However, we believe that it is important to understand the concept of "methodology" and look at other methodologies as well. For this purpose we have carried out the study work "XP versus UP – a methodology comparison study work". The work is presented in the Appendix A. It is not a complete or very thorough study work – this work is actually the first step we made in order to go higher above single UP methodology and try to understand the purport of different methodologies (namely XP and UP).

The study work highlights the main differences between UP and XP.

We qualified XP as the methodology best suited for small development teams (usually varying from 2 to 15 people), embracing change through fast iterations and feedbacks. Most notably XP is designed to be a lightweight methodology - that is XP focuses on following practices which makes software development go faster and remove practices which make it move slower (for example documentation).

UP, on the contrary, is designed for huge staff and big projects. A decent educational background is necessary to follow the process thoroughly. UP also embrace change through iterations, though because of bigger staff, the iteration naturally last longer than in XP.

The last most important note is that these two methodologies do not compete with each other – that is both methodologies stresses different parts of information system development and thus provide different approaches and practices.

Based on this information we can say that it is at least worth investigating and trying to research the possibility to blend these two methodologies. Rewebution system development team consists only of two members (XP trait). Project requirements are to use WEB-RUP methodology (a plus for UP). Having done risk analysis, we found that the biggest risk in our project is the web programming methodology and user interface design. XP is based on idea that the documentation is made only when it is needed, hence this trait could be used to investigate User Interface design and develop corresponding documentation. The other documents (from management activities and others), due to the size limitation of the project and project

scope, could be done only and only when they are necessary. XP also focuses on coding thus limiting design[1]; this could enable us to focus on web programming.

UP would contribute by providing the relevant set of artifacts and workflows, since XP is in most cases a collection of rules of thumb than a precise description of process.

This way the methodologies could be blended and help us in achieving our project's goals.

[1] The reason for this is thoroughly discussed in McConnell's paper published at http://www.amazon.com/exec/obidos/ASIN/1556159005

6 User participation and user interface design

In the inception phase our main goals were to understand the requirements and design the user interface for the system so later we could define the appropriate architecture and implement the system. Hence the user participation and user interface design will be main topics for this chapter.

6.1 Importance of user participation

The level of user participation varies greatly from project to project. However, many projects have the following attributes:

- **Systems are intended to achieve strategic businesses goals**
  This implies that the top management must define the systems boundary and the systems functionality, which will help in succeeding in business.

- **Systems are designed to be used by people**
  This implies that the systems users must be taken into account while designing the system. What might happen if users have little or no change control of the system they will use? It might happen that users could feel the system will make their job more demanding, or it could isolate and change their relationships with other co-workers. As a result of these feelings the user might express **aggression** towards the system by explicitly failing to use the system or lose documents. Users might blame the system for causing difficulties whilst the true facts are lying somewhere else. This is called as **projection.** The other users might choose **avoidance** tactics and try to avoid the system in all ways.

So the systems should be designed carefully in agreement with users or the system might fail after successfully implementing it. Sure there are always exceptions to the rules. This one also does and when the system is intended to work totally without any user interception (for example communicating only with other systems), the importance of user participation is diminished.

Rewebution system is highly sensible on good user interface design. Single user can be lost forever if the web page leaves bad impression; he will never return again. It is important to understand which level of user participation suits best while building the system.

6.2 Level of user participation

Mumford (1983b) distinguishes three levels of user participation:

- **Consultative participation**
  It is the lowest level of participation and most of the design tasks are worked out by the system analysts. However, all the staff in the user department is consulted about the change.
- **Representative participation**
  In this case, the design group consists of both system analysts and user representatives. Users have an equal say in any decision.
- **Consensus participation**
  This process is called user-driven. Though the decisions about system design might not be made as quick as could be, it has the merit of making the design decisions those of the staff as a whole.

It is possible to use other approaches for assessing the level of user participation.
The following assessment of user participation level was made by us to show how the user participation levels vary between system development stages. The evaluated project is using UP methodology to build a business application, designed to be deployed in user's organization.

Stage	Who to involve?	How would you involve these people?	Important results
Inception	Top management	Through discussion, and possibly workshops.	Planning and preparing a business case, a vision, Preparing the supporting environment for the project
Elaboration	Users of the system	Workshops, discussion, questionnaires, getting feedback from prototypes	Ensure requirements are stable enough (working with use cases) and review prototypes
Construction	Users of the system (but not as much as in elaboration phase)	Discussions , prototypes (possibly pilot projects)	Fleshing out the remaining issues, evaluate user interface
Transition	User, top management	through beta testing, testing the deliverable product at the end-user site	Validate the new system against user expectations, training of users and maintainers. Convincing top management the system is doing what it is supposed to do

6.3 User participation problems

Though user participation is encouraged to be used in building systems, it has also drawbacks.

- It might result in polarizing or fragmenting groups and choosing the 'right' users or by suggesting that users decide 'this… or there will be unhappy consequences'.
- Participation may cause resentment either from analyst, who might see that someone unskillful is overtaking his job, or by users, who feel that this is not of their business.

  Such lip service participation might result in the growth of "end-user computing". Or in others words – users start developing their own system. The other response would be joint requirements planning (JRP) and joint application design (JAD). Representatives from both the user community and computer people conduct workshops to progress the information system development.

6.4 Ways of improving user participation and involvement

There are many ways of improving user participation and involvement and all of them vary greatly. We will state only a couple of them here:

- **Training** of staff affected by computers, computer people should also be aware of business matters
- **Involve all affected by the system** into system development, these includes not only general staff, but also top-management. Involvement of top-management might act as a leadership by example to other employees.

- **Let user and analyst be in a working environment together** to encourage team-work and group knowledge – analysts have knowledge of analyzing systems whilst users are experts in the application data.
- **Improve the human-computer interface**. This includes:
  - **Visibility**. The users will feel more comfortable with the system if the system is to be more response sensitive.
  - **Simplicity** – the users must be able to understand and cope with information instinctively, without thorough analysis of "what did the system print on the screen…."
  - **Consistency** – This is sometimes referred also as "Look and feel". Essentially this means that the system must reuse human-computer interface patterns as often as possible.
  - **Flexibility** – It means that the users might adapt the interface for their needs.

Not all the ways are achieved easily, but these points form a definite guide to increase user participation and involvement.

6.5 User participation in Rewebution project

Rewebution system is a web-application and the users of this system will be scattered at least somewhere around Roskilde (because Rewebution system is build for Roskilde Grej). It is not like in other non-web projects, were the users of the system are known beforehand.

Probably we could choose one of the three Mumford defined user participation levels: either consultative, representative or consensus participation. Unfortunately, the problem with all approaches is the same – the users are not defined, the users are actually on the net. However, there is a specific approach for web-application development and is defined in WEB- RUP white paper [Web-RUP 1999]. The paper defines the necessary steps to be undertaken in order to create effective and simple user interface design. This is accomplished by firstly defining the guidelines for developing user interface (the Design Brief document) together with the Navigation Map document. Further, Design Elements document is used to stabilize the design guidelines. These documents gradually evolve user interface design step by step. We have added one additional procedure before stabilizing user interface with web design elements – the heuristic user interface evaluation is used to check the quality of the design and detect mistakes early enough, before more expensive system development activities start (such as building an HTML prototype).

6.6 Design Brief

Design brief is the initial user interface guidelines. These guidelines are used attempting to build the first user interface mock-ups (also called design comps or just simply comps).

6.6.1 Why provide a design brief?

The purpose of the design brief is to get everyone started with a common understanding of what's to be accomplished. It gives the guidelines for a variety of user interface topics: the mood of the site, the browser web application is expected to be designed for, colors, fonts and other.

6.6.2 Design Brief instances

We created design brief, which helped us to make design comps and later implement some use cases.
Below is an excerpt from our design brief which we wrote in our project. It provides some important constraints on the web application to be delivered.

- **The mood of the site**:
  It will be the service page for the rental company. The web page will be playful and giving service for users'. We have decided that playful page would be more attractive and attain more attention from the customers.
- **Browser**:
  We will make our web application compatible only with Microsoft Internet Explorer
- **Colours**:
  We will use few colours for web page (red, blue and black). It is suggested to use no more than three colours in web design.

Our group had interesting discussion about design brief. We talked about wherewith (design or requirements) design brief is related. We decided that it is obvious that design brief can be viewed as design or as requirements, it is only a matter of point of view. If the design brief was made by designers, then it is a design of the system, not the requirements for the system. If it was made by clients, design brief exposes requirements for the project. In our project, the design brief was made by us, acting as the designers, and thus it was a document highlighting the path for designing the system further.

6.7 Navigation map

Navigation Map is a view of the web application showing the specific presentation level pages in a hierarchical tree diagram. Navigation Map is also related to a functional design.

Each level of the diagram shows the number of clicks it takes to get to that screen/page. Generally, everybody wants to have the most important areas of the web site only one click away from the first page (commonly known as the "home page").

The development of the navigation map starts by identifying the major windows or Web pages for each of the use cases.

6.7.1 Why need Navigation map?

Making navigation map in the project provides better communication between stakeholders and development team. The users are able to better envision how the Web site will be navigated, and creative designers are better able to understand the necessary navigational scheme.

The navigation map naturally evolves from the use-case model, since the use-case model describes what services the system provides to the users. Concerning this reason, the navigation map should be created after the initial use-case model is finished.

6.7.2 Navigation map in our project

After design brief, we made navigation map for Roskilde Grej web page.

Navigation map can be made for different user groups or even actor specific if necessary. So, we could make three navigation maps: full site navigation map, for logged in users and not logged in users. However, we made only full site navigation map for all web site users.

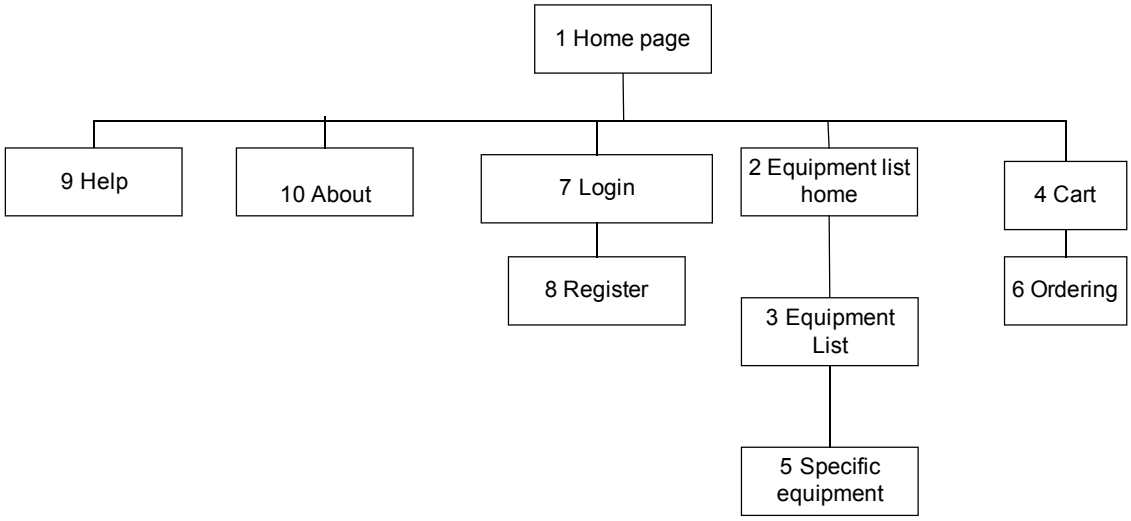


Diagram 1 – web site navigation map

Each level of the diagram shows the number of clicks it takes to get to that page/screen. (It will reflect later in design comps

6.8 Design Comps

Design comps consist of mock- ups of what the site might look like. These mock- ups are typically pictures, with a browser frame around it. The comps supply the interaction to the designer with a structural style guideline, and the customer with how web pages look like, before web application will be finished.

6.8.1 Purpose of design comps

The idea to create design comps is to give a look at our web site and get the feedback, which would help to understand mistakes and bad design early enough to resolve the problems easily. The problems we can find with the help of design comps might be from choosing inappropriate colors for web, wrong navigation between pages to even omitted system functionality.

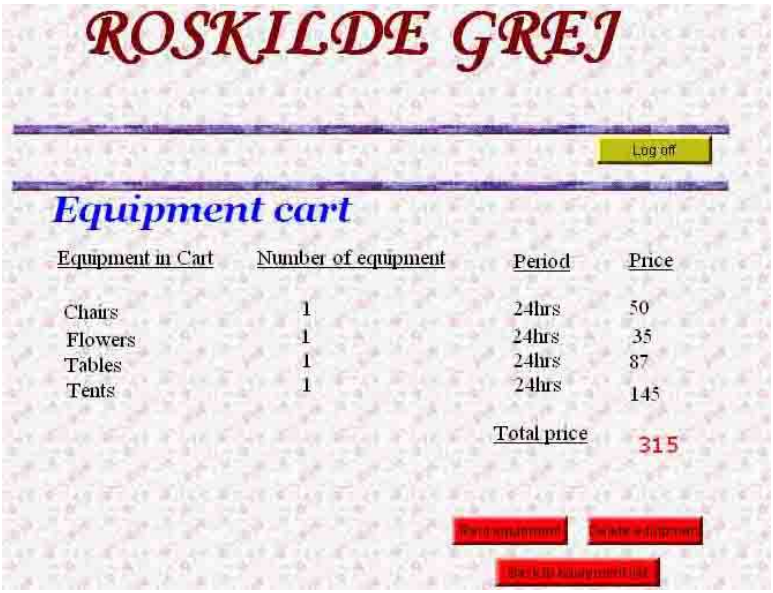
6.8.2 Design comps instances

Our design comps consist of 14 site pages. We created comps using our navigation map, and did not pay attention on comps colors, backgrounds, font size, which we considered being not so important in the beginning. Navigation between pages is important because in this way we have checked navigation map for consistency. Later we paid more attention on colors, font size, backgrounds, text and other design elements.

To create design comps, we had two opportunities: draw it on papers or use web building program. We did not draw comps on papers because it would be difficult to keep use them and understand and trace with the navigation map. However, it is not a bad solution, if there are only a few pages.

We chose "Web studio" web building tool. We have chosen it because it is easy to understand how to work with it. "Web studio" has a built-in ability to use links (which was very important because we wanted navigation between pages). In addition, we could put various web elements onto the page - backgrounds, buttons, links, pictures, animation, and text fields.

Here is an example of window "Cart" from our design comps:



Picture 1 – Cart window

6.9 User interface evaluation

There are basically four ways to evaluate a user interface: ***Formally*** by some analysis technique, ***automatically*** by a computerized procedure, ***empirically*** by experiments with test users, and ***heuristically*** by simply looking at the interface and passing judgment according to ones own opinion.

Formal analysis model is not applied in real software development projects yet.
Automatically evaluate user interface: is infeasible except for a few primitive checks.
Flow practice is to do empirical evaluation, if someone wants a good and thorough evaluation of a user interface. "Unfortunately, in most situations, people actually do not conduct empirical evaluations because they lack the time, expertise, inclination, or simply the tradition to do so"[1].

Heuristic evaluation is not a pure analysis technique. It might be thought of as "analysis by a team of analysers using a variety of informed models"[1]. A more descriptive term applied to heuristic evaluation is *Usability Inspection*: an inspection is carried out and a list of problems affecting usability is drawn up.

Here are some *advantages* for heuristic evaluation:

- It is cheap, because it is easy to understand, and use.
- It is intuitive and it is easy to motivate people to do it.
- It does not require advanced planning.
- It can be used early in the development process.

A disadvantage of the method is that it sometimes identifies usability problems without providing direct suggestions to solve them. The method is biased by the current mindset of the evaluators and normally does not generate breakthroughs in the evaluated design.

[1] Heuristic Evaluation prepared by Rajesh Vijayan

We did heuristic evaluation in our project to identify possible flaws in design comps. The comps were evaluated on the *ten general principles for user interface design*[1]. They are called "heuristics" because they are more in the nature of rules of thumb than specific usability guidelines. We present here the most important aspects of this evaluation.

- ***Visibility of system status*** - the system always informs users about what is going on.
- ***Match between system and the real world*** – the system always speaks the users' language.
- ***User control and freedom***
- ***Consistency and standard****;*

We have conducted a heuristic evaluation and ensured that above and other six heuristic are applied consistently throughout the website in all design comps. Now we are ready to define the graphic standards for our Rewebution system by identifying important parts in our design comps.

6.10Web Design Elements

Web design elements are the discrete graphical images, which are used to build the Web pages for a site. Examples of web design elements include graphical elements such as navigation devices, fonts, text, page backgrounds, logo's, etc.

6.10.1The purpose of web design elements

Web design elements should help create and stabilize web page design. Consistency of the user interface across a web-site should provide a consistent user experience. To ensure this, the project must use a set of standard graphical components.

6.10.2Web Design Elements in our project

Here is the most important web elements defined in our project.

Font: the pages should be written in Times New Roman and Verdana fonts.
Font size: 12 pt font sizes should be used consistently across the web site. Headings should use 18 pt font sizes.
Text colors: for ordinary text black color should be used. For headings and highlighted items blue or red.
Background: the following table summarizes the use of backgrounds

This background is for main page only	
The other pages should use this background	

6.11User interface description

The last thing we did to increase and ensure the common understanding of the user interface was the User Interface description document.

[1] http://www.useit.com/papers/heuristic/heuristic_list.html - by Jacob Nielsen.

This documents goal is to strictly define the functions of the window together with the important components inside. This document refers primarily to use case model document and so the overall and deep overview of the system can be comprehend thoroughly by studying these two documents.
Here we present the excerpts from these documents to illustrate how the login window was designed.

Use case model document – UC114 Log in example

13. UC114 Log in
Short description: logs in customer
13.1 References
Concrete and base use case
Included by - none
13.2 Flow Of events
13.2.1 Basic Flow:
1. The system asks user to type in user name and password
2. Displays the link to UC114 register new customer , if user has no account
3. The system verifies that user name and password are correct and logs in customer
13.3 User interface
WND7 Login

Navigation Map document – WND7 Login

1. WND7 Login

1.1 Description

This page must be reachable from all pages. See list of all reachable pages:
Corresponding use case - UC114 Log in
Previous window: WND1 Home Page
Next Windows: WND1 Home page, WND8 Register User

1.2 Objects and actions

- Text box – user name
- Text box – user password
- Link – New User
  WND8 Register User
- Link – Submit
  WND1 Home page
- Link - Cancel
  WND1 Home page

1.4 Screenshot Sample

Picture 2 – Login window

7 Client-Server architecture

In elaboration phase iteration 1 we were focusing on defining system's architecture, thus the following sections will mainly discuss architecture both in theoretical way and how we applied it practically in our Rewebution system.

7.1 Introduction to client-server architecture

Rewebution is a web application. Network, and more especially internet and WWW (World Wide Web) technology is very young (not even 20 years have passed since the dawn of the WWW, therefore many different solutions are made to the same problems. During this project we tried to summarize and elicit the most common solutions for web applications regarding client-server architecture.

We start by defining what a web application is, then analyze the three client-server architecture patterns and present the architecture design applied in Rewebution system.

7.2 Definition of the Web Application

Web applications grew out of web sites. Essentially, a web application is an extended web site allowing its users to invoke business logic and change the state of the business state on the server.
This definition implies that a web application must have at least three parts:

- A client browser
- An application server
- A web server

This definition does not limit web application, nor does it reject the use of distributed objects, Java applets or use of other technology. The purpose of this definition is to formulate a common background for discussing client-server architectural patterns.

7.3 Three common used web-application architectural patterns

Having defined web application as a system composed of 3 most important parts (browser, web server and application server) we have chosen to highlight 3 possible architecture patterns. These are

- **Thin Web Client**
  Suited mainly for e-commerce projects targeting as much customers as possible. All business logic is executed on the server. The connection between server and client is carried through HTTP connectionless protocol.
- **Thick Web Client**
  Architecturally significant amount of business logic is executed on the client machine. Communication is still carried through HTTP. This pattern exposes some limitations to client browsers.
- **Web Delivery**
  In addition to the use of HTTP for client communication with the server, other protocols may be used to support the distributed object system. In this case, web browser acts as a delivery and container device for a distributed object system.

In the following sections we present the common components to all of the patterns. Next we will examine each of the patterns individually and point out the main differences and consequences of using them.

7.3.1 Core components

- **Browser**
  Displays the result acquired from the web server
- **HTTP**
  A connection protocol used to link the browser and the web server
- **Web server**
  Simply fetches html pages and returns to the browser. If the requested pages are server page and include scripting elements, the pages are forwarded to application server and the result is sent back to the browser.
- **Application server**
  Executes the code in the server pages
- **HTML pages**
  A web page with user interface and content information
- **Server pages**
  A web page containing code. Usually code is written in ASP, PHP, JSP, Cold fusion or other languages. These pages have access to all server-side resources including business logic components, databases, legacy systems, and merchant account systems.

7.3.2 Common additional components

Though the core components may be sufficient for simple projects (as to what is referred as "brochure web site" in business terms [e-commerce]), others might need to run sophisticated business logic. The components are as follows:

- **Business objects**.
  Encapsulates the business logic, usually runs on the application server. Without this layer, the server pages should access database and/or other server-side resources directly and thus would lead to high coupling.
- **Persistence**
  Web applications might use different technologies to make business objects persistent and this component hides the details of the chosen technology. It might also include the use of a transaction-processing monitor (TPM).
- **Persistence mapping**
  Relational databases are the most common nowadays, so this component maps objects to relational database tables.
- **Merchant account system**
  This component allows web application to process credit card payments.
- **Legacy system interface**
  This component provides a common way to access legacy systems (such as an accounting system or various scheduling systems).

7.4 Thin Web Client

7.4.1 Application domain

This pattern is most appropriate for Internet-based web applications. It is also perfect in use for e-commerce business, brochure web sites, since this pattern assumes client has no or little control over its configuration and offers standard user interface support by probably all browsers.

7.4.2 Logic

The main idea of this pattern is that business logic is only changed through the requests of the client. No change is made after the client request has been executed.
It will perfectly utilize the HTTP protocol for establishing connection between client and server. No permanent connections are used. The client is limited to simple user interface and the business logic should be executed in a time frame expected by the user and users' browser.

7.4.3 Logical view

The client requests a resource (a web page). The web server either fetches the html page or server page. If it is a server page, the application server executes the code inside the server page. The web server returns the generated HTML output to the client. The server page can interact with various server-side resources through business layer or directly accessing persistence component (or with the help of persistence mapping).

The following diagram presents the logical view of the Thin Web Client.

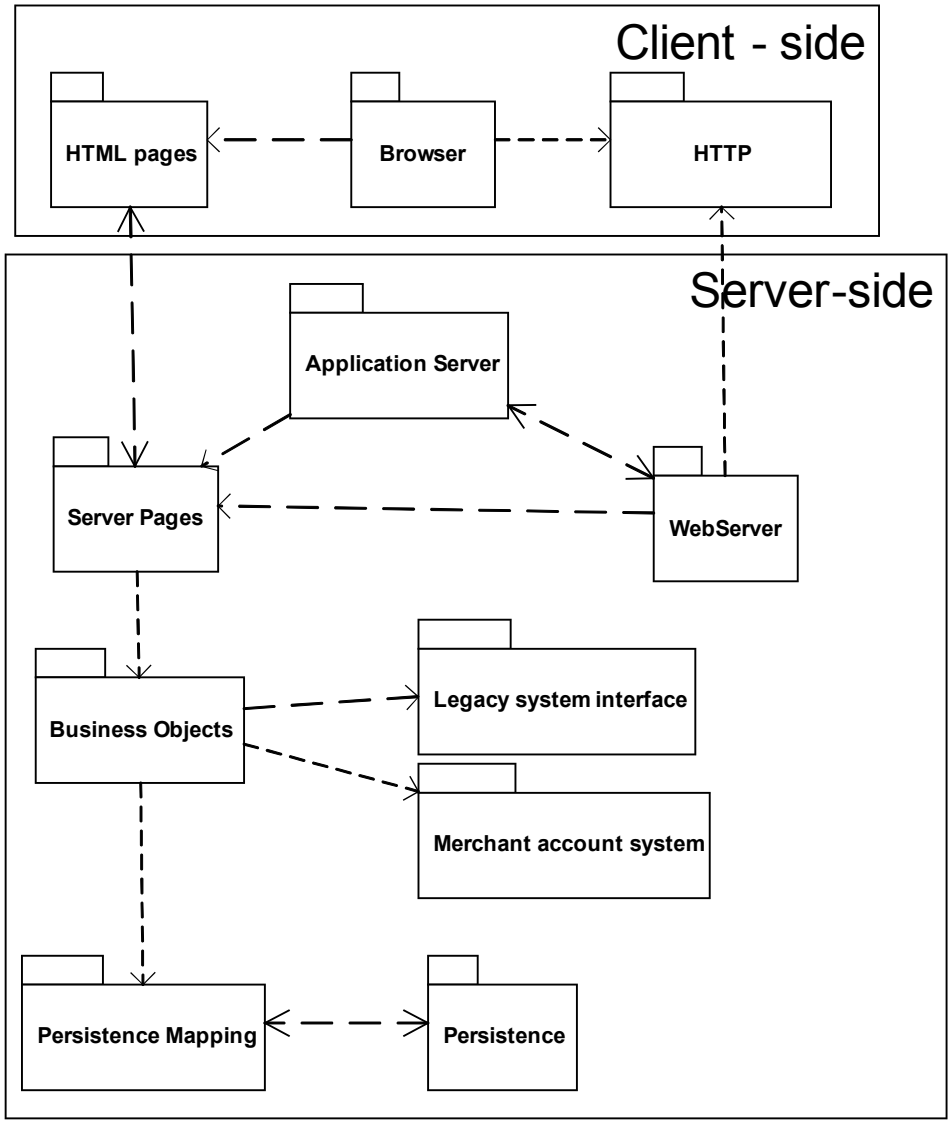


Diagram 2 – logical view of Thin Web Client [1]

The logical view was divided into two separate parts – client side and server side to stress that the business logic is executed on the server (after all, all business objects reside on the server-side).

HTML pages were considered to be on the client-server as to show that they are not leveraging the business process. On the contrary, server pages do change the business process and were put on server-side.

[1] This and other diagrams about client-server architecture were taken from the book ”Building Web Applications with UML” and changed to suit our needs.

7.5 Thick Web Client

7.5.1 Application Domain

Pattern suits where more sophisticated user interface needed or a certain amount of business logic might be executed on client-side. User configurations must be more concrete and specific to meet the web application requirements.

7.5.2 Logic

This pattern generally speaking extends the Thin Web Client by the use of various scripts processed by the browser (JavaScript and VBScript) and various objects like ActiveX controls and Java applets. Unfortunately not all browsers support scripts and there are differences in implementations of the Document Object Model [1](DOM).
XML documents can be used to store data without user interface formatting.

7.5.3 Logical View

The flow of events is the same as in for Thin Web Client except the browser is to execute the client-side scripts. At the same time, Java applets or ActiveX controls can execute and access client-side resources.
The logical view mostly the same. It is only client-side which has changed.

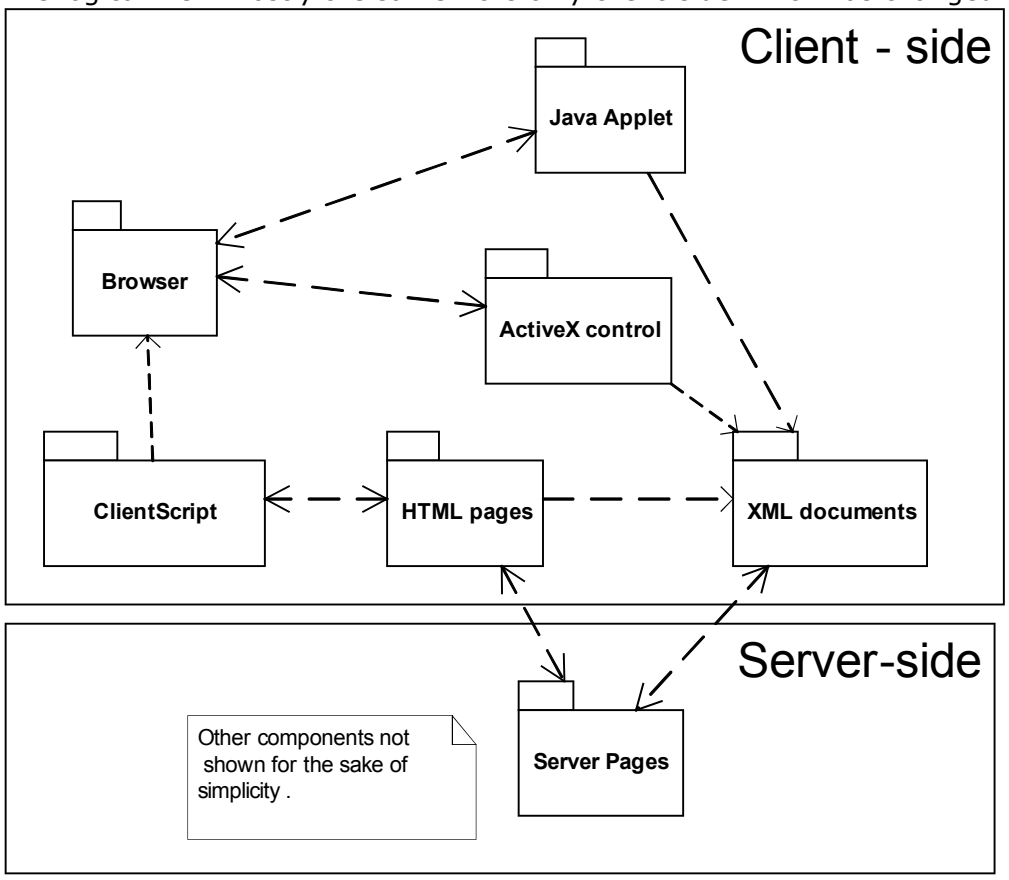


Diagram 3 – Thick Web Client

[1] DOM is a W3C standard for giving scripts, controls, and applets access to the browser and HTML content in pages.

7.6 Web Delivery

7.6.1 Application Domain

This pattern is best suited where a significant amount of business logic is advantageous to execute on client side. Sophisticated user interfaces greatly extending standard HTML forms can be achieved.

7.6.2 Logic

The Web Delivery architectural pattern might be viewed as a type of application where distributed object system happens to use Web elements (or vice versa).
The browser is used only to deliver a distributed object system. The advantage to use a browser is that browser has an in-built capability of automatically downloading and installing necessary components to run the application. All what new computers need to have to run the application are a browser and an Internet connection.
The pattern is not considered to be used alone in isolation. Typical application would use thin or thick web client for parts of the system not requiring sophisticated user interface.
No longer is connection between server and client made only through HTTP connectionless protocol. DCOM[1], IIOP[2], or RMI[3] protocols are used to implement interactive actions between user and server.

7.6.3 Logical View

Web delivery further upgrades the pattern by letting Java applets access server-side resources through RMI or IIOP protocols and ActiveX components through DCOM. This results in the following logical view of the pattern

[1] DCOM: Distributed COM is Microsoft's distributed object protocol for enabling interaction and invocation of methods on objects on another machine
[2] IIOP: Internet Inter-ORB Protocol is OMG's CORBA protocol for interacting with distributed objects across the Internet or any TCP/IP based network.
[3] RMI: Remote method invocation is the Java way of interacting with objects on other machines. Could be implemented with JRMP (Java Remote Method Protocol) or CORBA's IIOP.

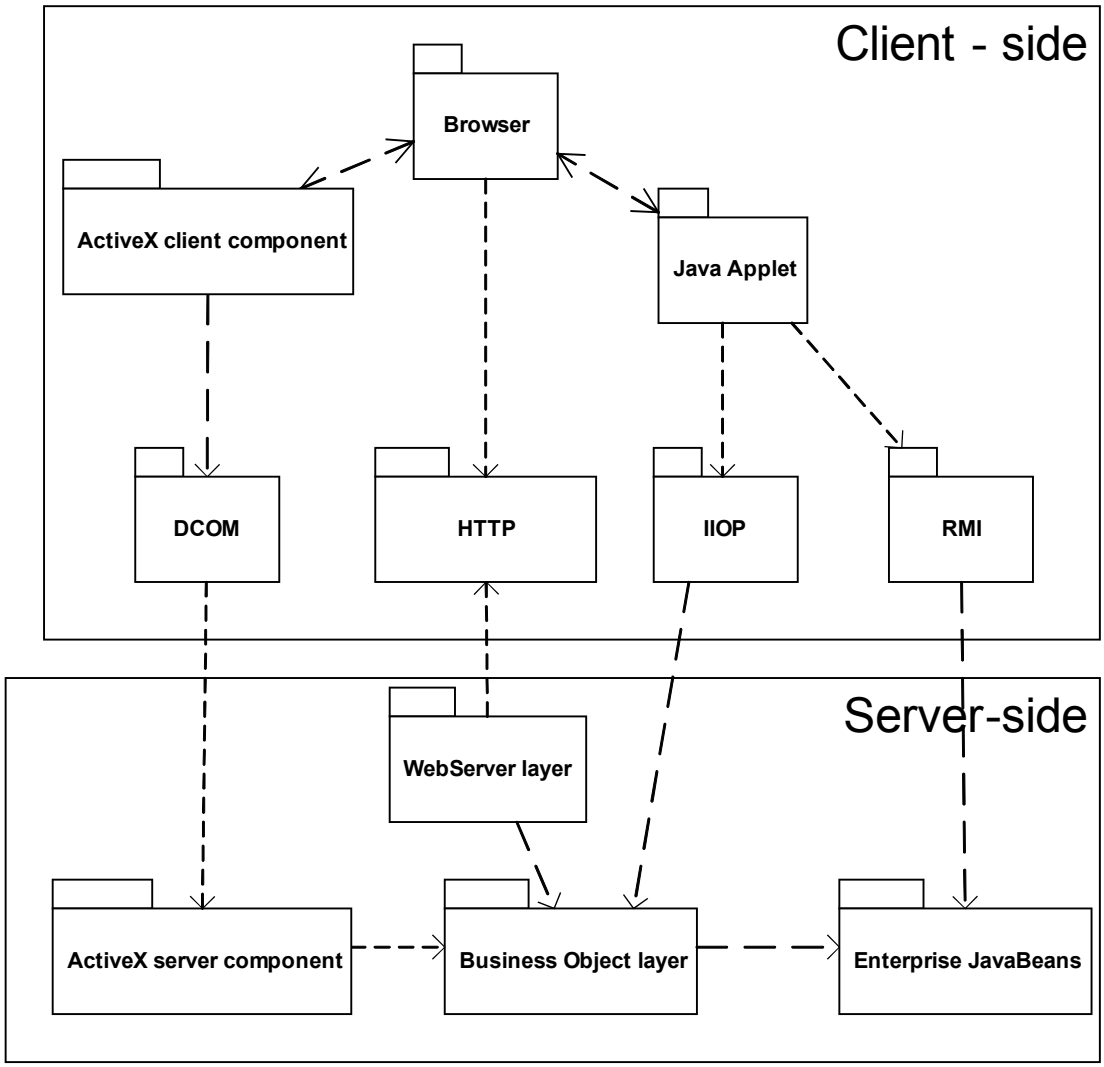


Diagram 4 - Web delivery architectural pattern

7.7 Overview of the three architectural patterns.

The following table summarizes and highlights the attributes of the three architectural patterns.

Attribute	Thin Web Client	Thick Web Client	Web delivery
Role of the browser	Provides standard user interface	Standard user interface and container for custom components	Delivers distributed object system (container for distributed objects)
Connection protocols used	HTTP	HTTP	Various – HTTP, JRMP, IIOP, DCOM
Place of business logic execution	Only server-side	Mainly server, but various custom components are allowed to leverage business logic on client-side.	Mostly client-side through distributed objects; however, this pattern is usually used together with thick and thin web client patterns

Used main technologies	HTML, HTTP	HTML, HTTP, client scripts (Java Script, VBScript), Java applet, ActiveX control, XML	RMI with JRMP or IIOP, Enterprise JavaBeans, ActiveX components with DCOM
Client configurations expected	Any browser complying HTML specification. Probably all users could run the application having a simple browser.	Browsers capable of executing client scripts and/or ActiveX controls (users are restricted to use specific technologies)	Configurations must suite very strict application needs

Table 1 - architectural pattern comparison

This table by no means is mean to be understood as strict and in all cases correct. The table should highlight the main attributes of patterns and provide the basis to understand the concepts. It does not restrict or limits patterns. It is only an attempt to summarize leaving out specific details.

7.8 Content management systems

So far we have covered three web application architectural patterns: thin web client, thick web client and web delivery. There are many more patterns for web applications. One of them is called Content Management Systems, or shorthand syntax CMS[1]. We believe that this pattern is important enough to get acquainted with it in this project.

CMS are basically designed to support the rapid change of web application contents (as can be easily understood from the pattern name). The main target of CMS is news-providers sites, web-blobs or even poetry or photo hosting sites – the sites with many similar designed pages though having different contents. There are many examples of such applications http://www.delfi.lt, http://rasyk.kitoks.lt, more. (Sorry, most examples are in lithuanian).

7.9 KillBenas web application with CMS

The logic behind the CMS pattern is that the core of the web-application is only one single page and that page uses the database to store and retrieve other pages. Here we are going to present DarkACM system developed by one of our members. The system was used to develop KillBenas web application.

The web application was developed a few years ago. PHP scripting language was used to implement the system.
The following diagrams show the logical view of KillBenas system.

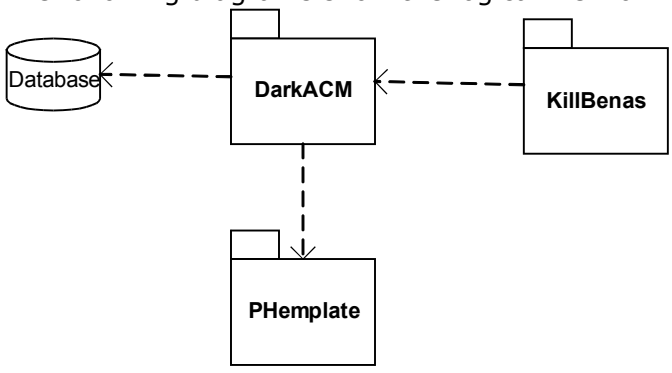


Diagram 5 – KillBenas logical view

[1] We have found also these systems being called ACM systems – Advanced Content Management systems.

The diagram is not meant to explain how the KillBenas web application works, but to highlight the architecture of the system – that is the use of the content management system (DarkACM) and template engine (Phemplate). The system actually has much more components (Language, Viewer, Installation etc.), but we are not here to present the KillBenas itself. The highlighted components play the following roles:

- **PHemplate**
  This is the template engine component developed by Juozas Šalna (http://pukomuko.esu.lt). It lets separate PHP code and HTML code. This is done by writing html template for a page and then letting template engine to parse templated page and filling it with data generated with PHP script.
- **DarkACM**
  The core component of DarkACM system is used to retrieve pages from database. DarkACM component after retrieving pages from database builds the page by calling Phemplate component functions to combine the html page template and PHP script generated data.
- **KillBenas**
  The component that actually uses the CMS.

The next diagram shows KillBenas web application interaction diagram.

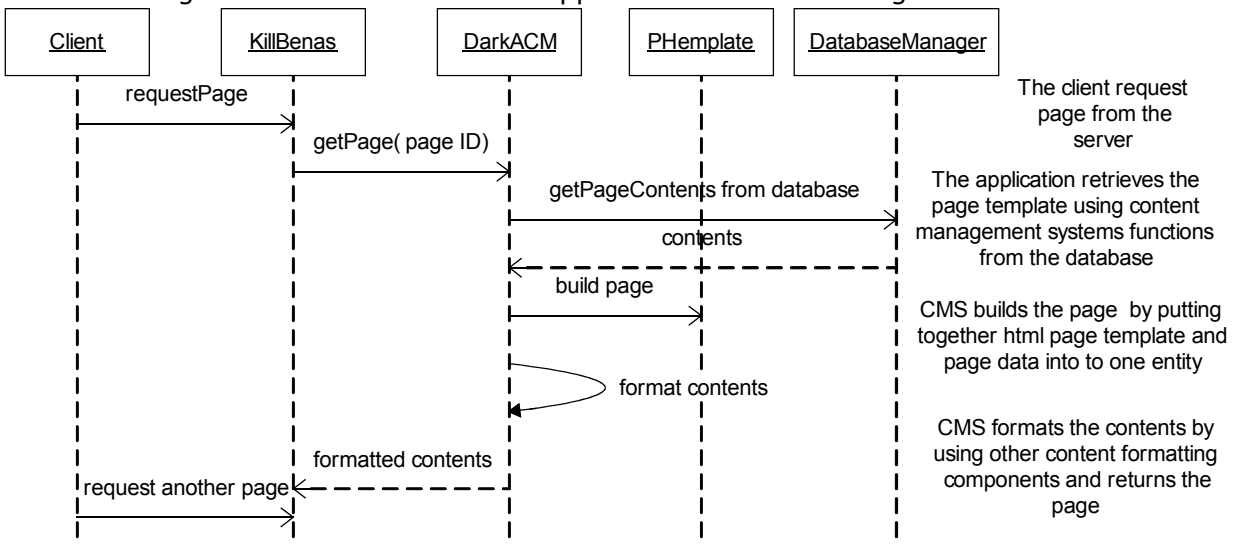


Diagram 6 – KillBenas sequence diagram.

The following code example shows how short and simple is the use of DarkACM system to provide content management functionality to web application.

```
// include DarkACM and Phemplate components
require_once('core/darkcms.class.php');
require_once('core/phemplate/phemplate.class.php');

// initialize phemplate, set display language
$tpl=new phemplate('core/templates/','keep');
$tpl->set_var('lng',$lng);

// initialize DarkACM component
$darkcms=new DarkCms();
// include and initialize viewer component
require_once("core/viewer.class.php");
$viewer=new viewer($darkcms,$tpl);

// generate the output
$output=$viewer->render($action);
$tpl->process('contents',$output,1,0,1);
```

```
// print the output
echo $tpl->process('Main','contents',1,0,1);
```

Code example 1 – KillBenas system

7.10Architectural pattern of Rewebution System

One of the goals of the elaboration 1st iteration was to define the architecture of the Rewebution System.

Roskilde Grej is relatively small company with little staff. Currently, they have Rental System and no steps are made towards doing e-commerce. Rewebution will be the first attempt to enter e-commerce sphere.
Having this information it is clear that the application will be moderate and no sophisticated user interface needed. What is more, since Roskilde Grej goal is to reach as many as possible customers, no specific user configurations should be considered. This led us to the final decision – the web application architectural pattern will be the Thin Web Client.

7.11Designing the system

Because of the lack of experience working with MS-Visio diagramming tool, we were not able to draw the appropriate shapes representing server and client side pages. Thus the following shapes will be used instead.

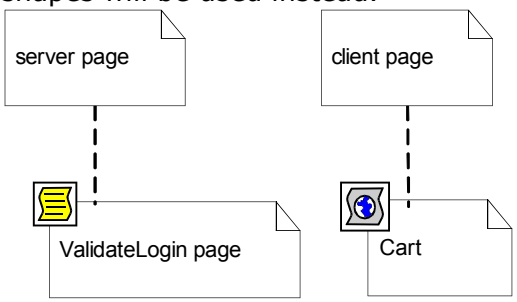


Diagram 7 – server/client page notation

7.11.1 UC102 View Equipment List Home

Having chosen the pattern, we started partitioning analysis objects into server and client side. This was done quite straightforward since all of the business logic goes on the server. All the data processing was put in server pages and the output (HTML pages) were put to client pages.

Interaction Diagram

For the use case **UC102 View Equipment list home** the following Interaction diagram was drawn:

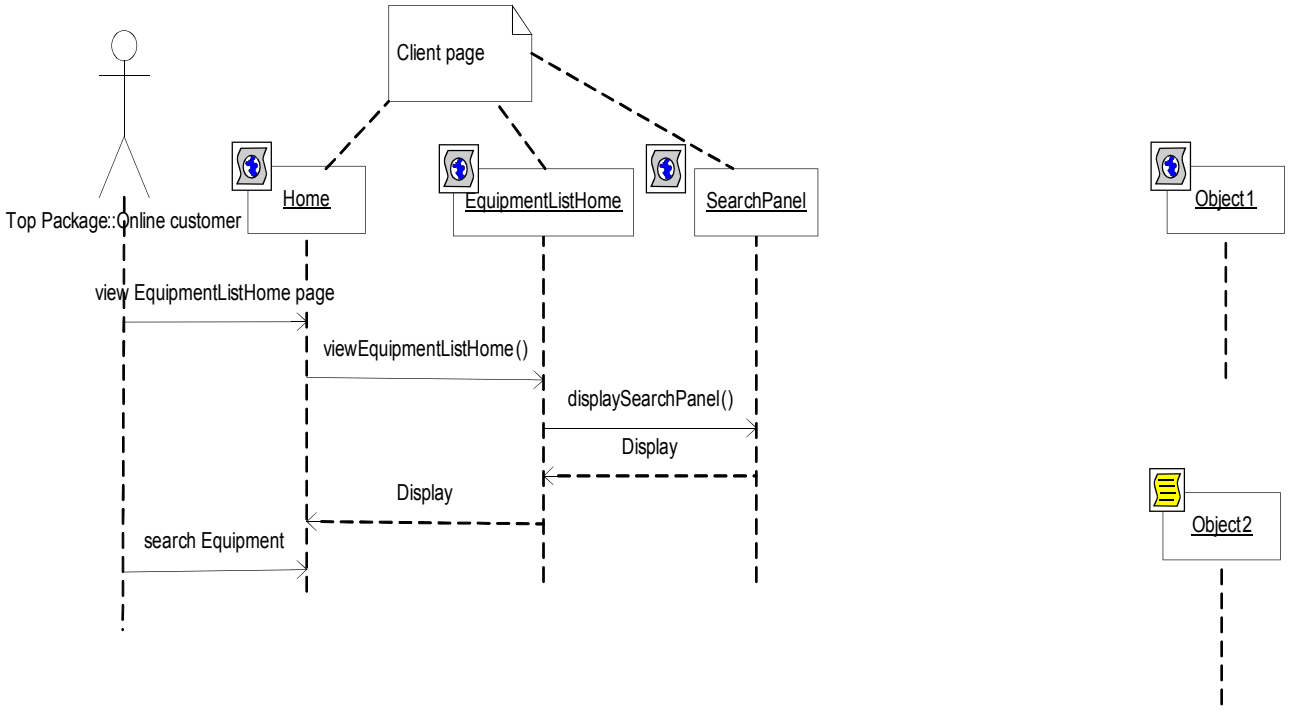


Diagram 8 - View Equipment List home Interaction diagram

The message ViewEquipmentListHome() would probably be implemented by putting a link to another page with an anchor tag (<a href="equipmentListHome.jsp") and displaySearchPanel () by including the HTML contents.

Logical View

The whole use case is very simple and realized straightforwardly collaborating java server pages. The next diagram shows the logical pages view realizing this use case. The pages of concern are EquipmentListHome and SearchPanel. Other pages are included for clarification purposes.

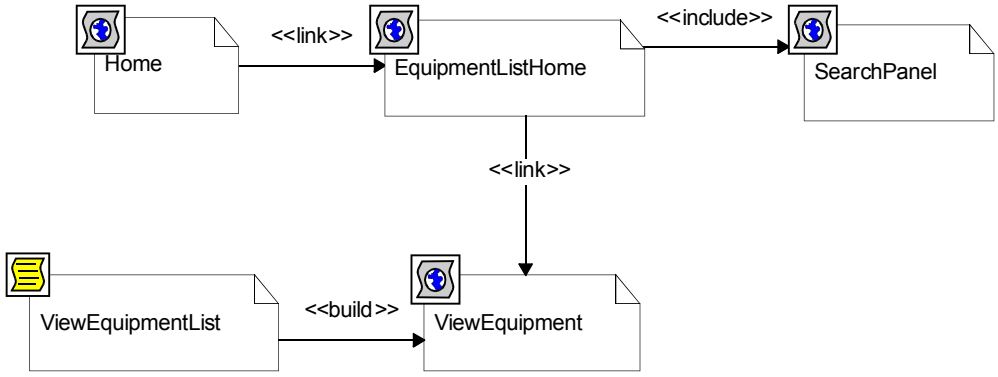


Diagram 9 – Logical pages view realizing UC102 View Equipment List Home

Components realization

The next diagram shows the component realization of this use case. (The page ViewEquipment was excluded since it is part of another use case and the diagram would become overburdened. It will be shown in the next example).

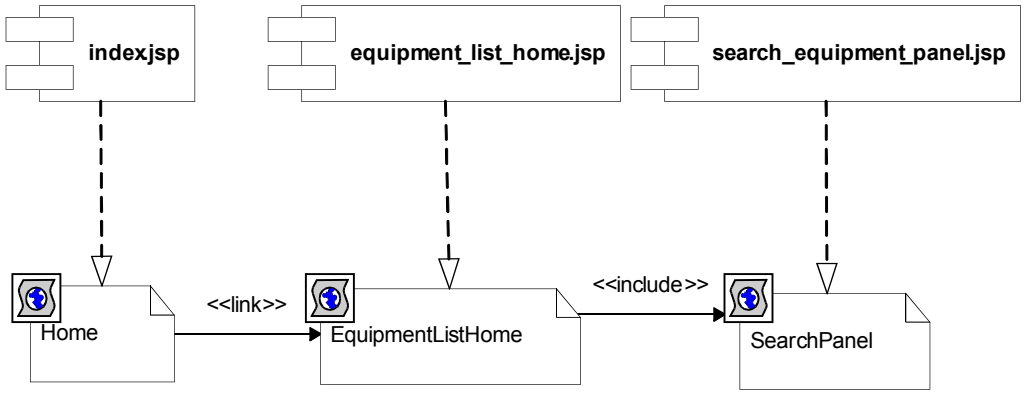


Diagram 10 – Component realization of UC102 View Equipment List Home

7.11.2 UC107 Put Equipment To Cart

The other use case **UC107 Put equipment to cart** was much more complicated and in designing interaction diagram we considered to use java bean to implement the cart. This design decision was made because beans help separate code from user interface and achieve better system scalability.

Interaction Diagram

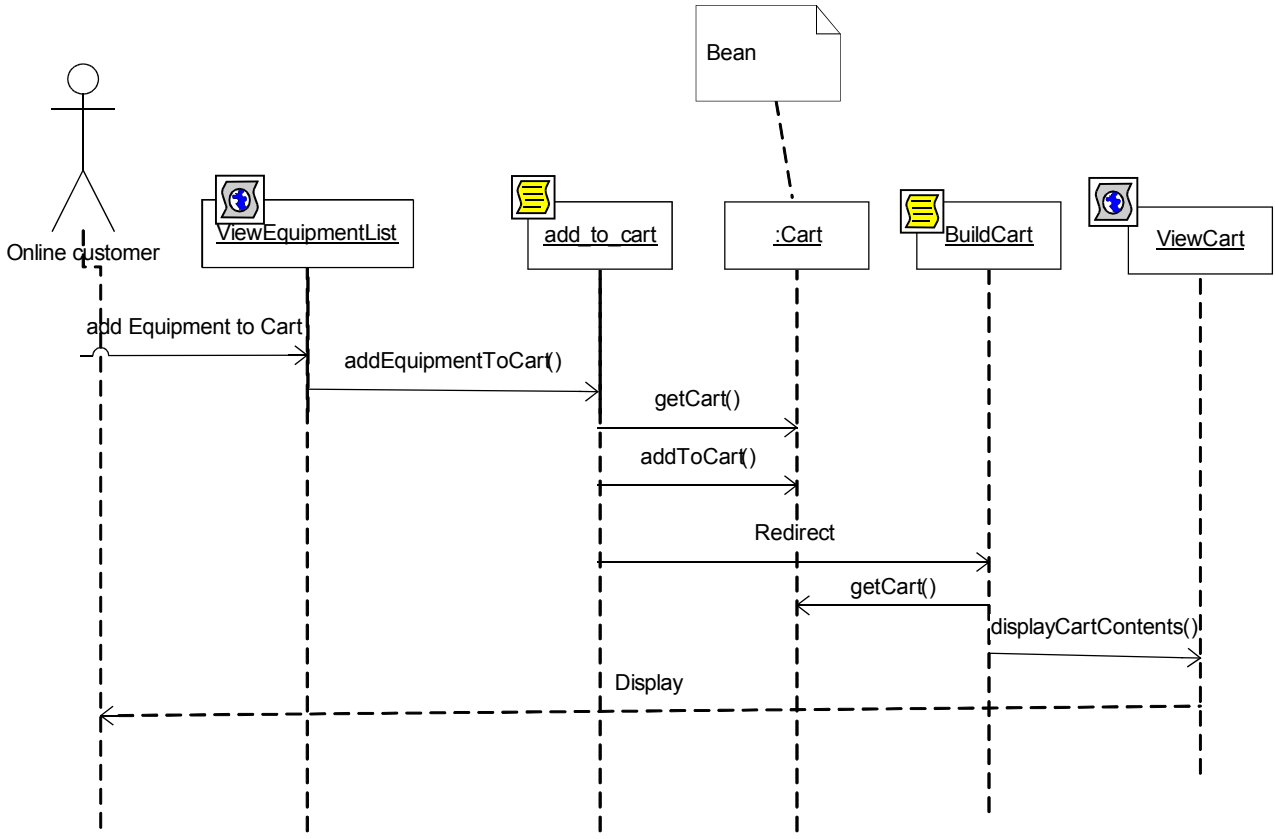


Diagram 11 - UC107 Put equipment to cart interaction diagram

Logical View

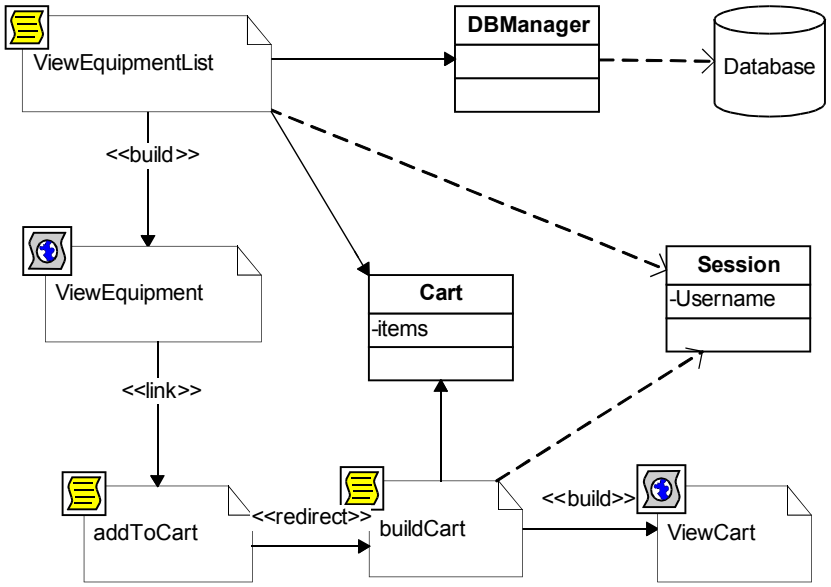


Diagram 12 – Logical view UC 107 Put equipment to cart

The link from ViewEquipmentList server page to View equipment client-page stereotyped with <<build>> tag actually does not represent the best solution for this situation. The reason is that it does not follow the Model View Controller approach (/architecture), thoroughly discussed in the subsequent Web programming section. The idea is that server-pages should not build the client pages, but forward to other pages without server-scripting elements which would generate the client-page. So in this case, the stereotype <<build>> is not reasonable. Stereotype <<forward>> would fit here more.

The next diagram clearly shows how we implemented this use case with four components.

Components realization

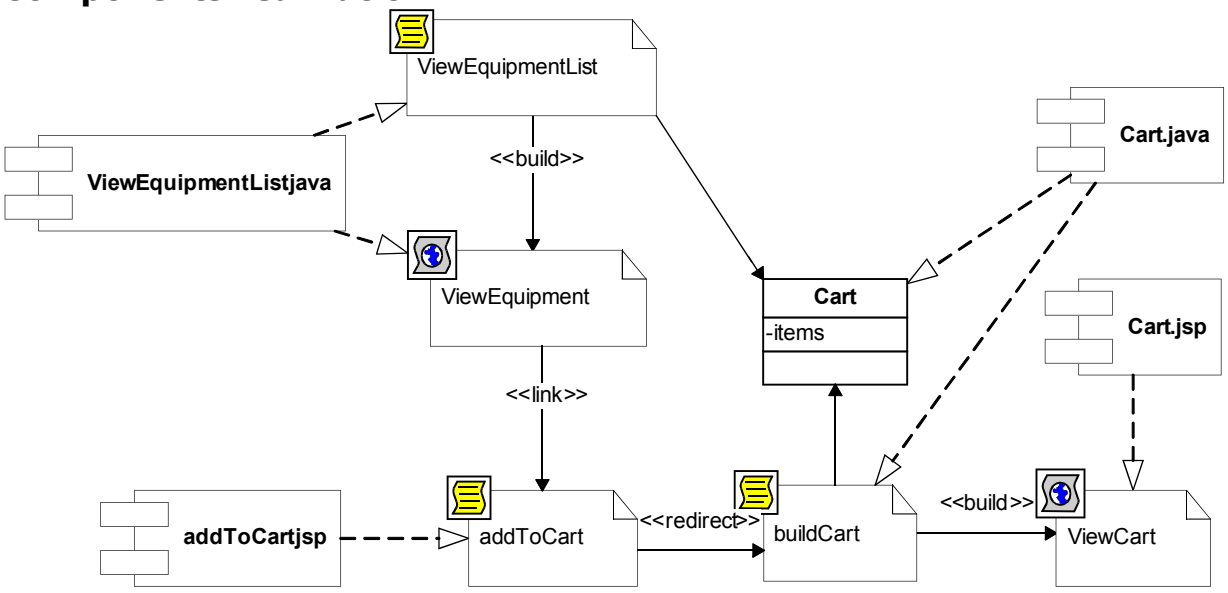


Diagram 13 – Component realization UC 107 Put equipment to cart

8 Web programming

In Elaboration phase 2nd Iteration our main goal was to implement a few use cases, hence our main topic will be the web programming.

8.1 Introduction to web programming

Web programming at the first glance is rather confusing topic, especially if encountered for the first time. Rewebution system is due to be implemented with Java Servlet and Java Server Pages technologies. Java Beans (not Enterprise JavaBeans) are also to be used where appropriate. In the subsequent sections we are to overview different approaches to implement web application. The area of interest will be writing java server pages, servlets, JavaBeans and RMI technology. We discuss HTML altogether, as it is the basis for all of the browser and web applications.

8.2 HTML

HTML is standard language used by all web browsers. Hyper Text Markup Language(HTML) is basically a collection of tags which can be grouped together to gain different results.
While we will not talk about the varieties of different tags, we will discuss one important tag <form>.
The <form> tag basically is used to enclose various graphical widgets such as buttons, input boxes, and check boxes into one logical section. In addition to this, it also defines which method to use to send information when submitting information from one page to another. There are two methods to submit data – either using POST or GET method. The POST method sends the data to the server in the body of the request, rather than as a query string appended to an URL (as in the GET method).

```
// create the web form and send the results to ViewEquipmentList servlet
// use the post method to send data
<form action="../servlet/ViewEquipmentList" type="post">
      // define some text boxes
      Equipment type <input TYPE="textbox" name="type" VALUE=””> <BR>
      Equipment category  <input TYPE="textbox" name="category" VALUE=””> <BR>
      Equipment price  <input TYPE="textbox" name="price" VALUE=””> <BR>
      Equipment year <input TYPE="textbox" name="year" VALUE=””> <BR>
      // create the submit button
      <INPUT TYPE=submit name=submit Value="search">
</FORM>
```

Code example 2 – creating html forms

The web page after receiving submit request from the user, sends the data to the ViewEquipmentList servlet. What is a servlet we are going to discuss just in the subsequent sections, but before that we will introduce java component models and accessible objects to JSP and servlets.

8.3 Java component models

In this context, a component model defines a set of contracts between component developer and the system which hosts the component. The contracts define stipulates how a component should be developed and packaged. There are a number of Java defined component models. These are: Enterprise Java Beans (EJB), servlets, and JSP pages (applets also considered to be java component model, though we will not discuss them here).
In the following sections we will look at each of the component in more detail.

8.4 Accessible objects to JSP and servlets

Both JSP and Java servlet have access to the information sent to and from the browser. The following class instances are accessible to JSP and servlets.

1. the **HttpServletRequest**
   Allows getting request data sent to server such as request parameters sent either using POST or GET methods.
2. the **HttpServletResponse**
   Represents the response data to send to the client: what type of document is being returned, cookies and other.
3. **PrintWriter**
   Provides a way to build the document, which is sent back to the client (together with the response).
4. the **HttpSession**
   Provides a way to identify a user across more than one page request and to store information about that user.
5. **ServletContext**.
   This is a data structure shared by all servlets and JSP pages in the web application.

8.5 Java servlet defined

A servlet is a Java class that is designed to respond with dynamic content to client requests over a network. Common Gateway Interface (CGI) is an international standard to write web components. CGI components can be written in any language one likes. A servlet is Java technology which can replace CGI.
Java servlet is executed within a runtime environment provided by a servlet container or web container such as Jakarta Tomcat. The container loads and instantiates all servlets before starting the web application. Servlets are parts of web application along with many other resources (images, html files and so on).

Concerning Rewebution system we have created a servlet for validating user login information (Use case id - UC114 Login). Unfortunately, logging user onto the system does not follow the MVC architecture (which will be discussed shortly in the upcoming sections). The ValidateLogin servlet checks if the users entered login information is correct and builds the output. Following the MVC architecture, the servlet should forward the output building to a JSP page so that view would be put apart from the code.

The following diagram illustrates the logical view of pages related to login.

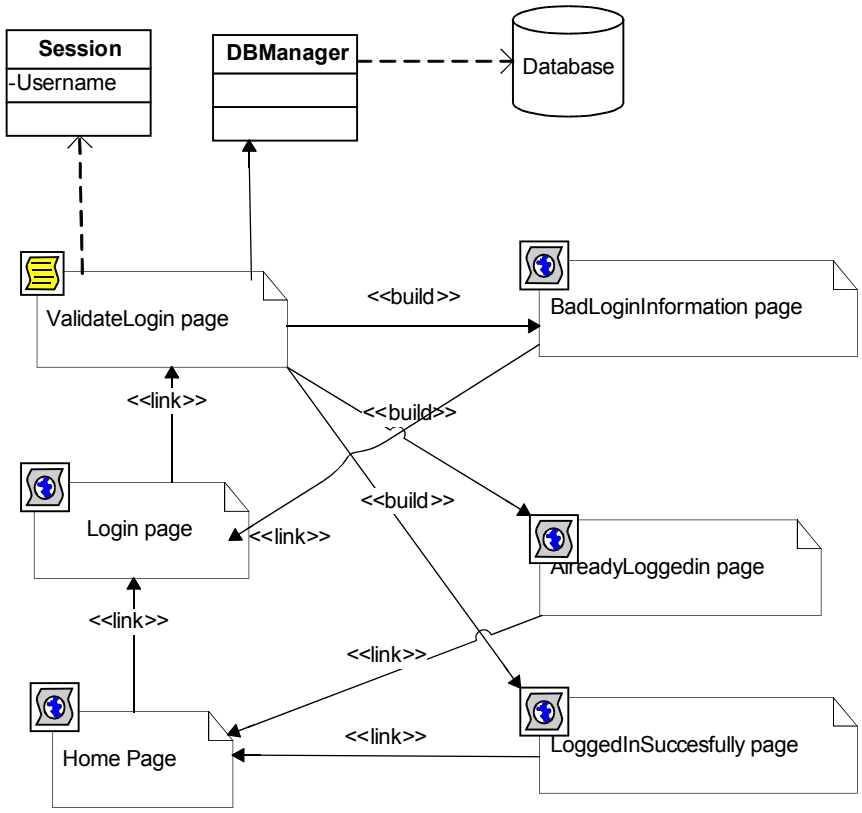


Diagram 14 – Logical view pages related to login

The DBManager class implements the Persistence Layer in our Thin Web Client architecture. It hides the details of specific database system used.

The next diagram shows the component realizations of login pages. The component "login.jsp" builds login page, while ValidateLogin servlet component builds ValidateLogin server side page and AlreadyLoggedIn, LoggedInSuccessfully, and BadLoginInformation client pages.

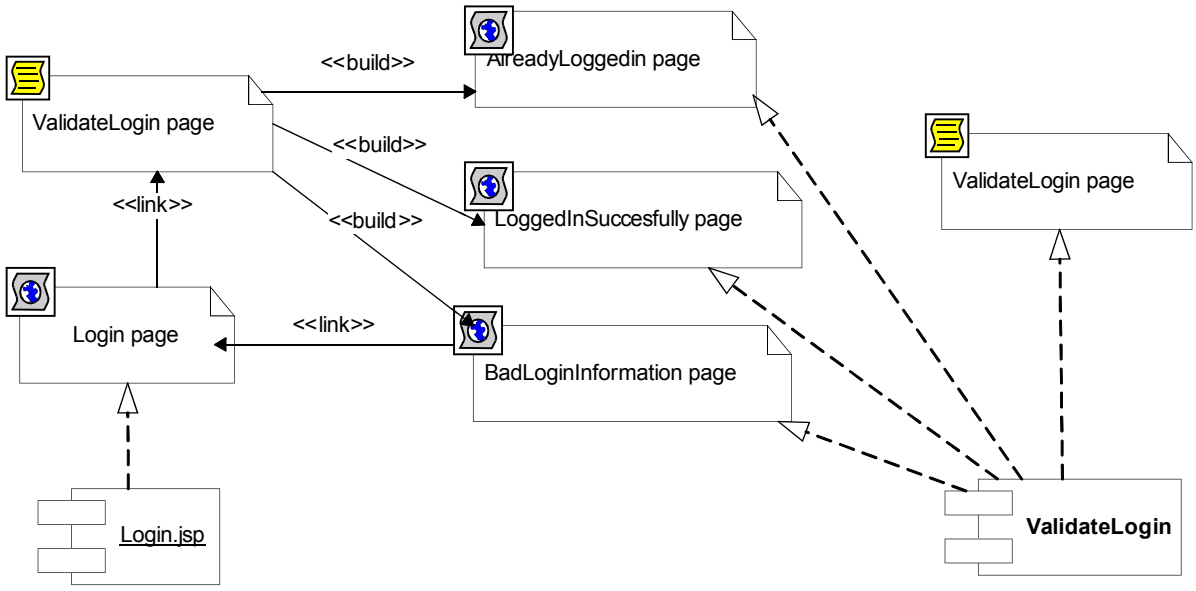


Diagram 15 – component realizations of login pages

And finally, based on this design we have implemented a servlet. A servlet is no more than just a normal Java Class (as happens with everything in Java world☺) extending a specific class. We have chosen to create a servlet by extending an HttpServlet class. Here is the source code, which is rather shortened to leave only the most important details.

```
//Creating a servlet class
public class ValidateLogin extends HttpServlet
{
      //defining write for sending html into
      PrintWriter out=null;

//    returns true if the password and username supplied are correct
      private boolean validateLogin(String username,String password)
      {
            // get the persistence object
            // DBManager was implemented as a Singleton class, which means
            // there is only one object instance of this clas.
            DBManager db=DBManager.getInstance();
             // create a query to get password for the supplied loginname
            String query="select PASS from CUSTOMER where LOGINNAME = '"+user-
name+"'";
            //execute the query and put results in ResultSet
            ResultSet r=db.executeQuery(query);
            String passw="";
            try
            {
                  //move to the first row
                  r.next();
                  //get the password
                  passw= r.getString("pass");
            }
            catch (Exception e)
            {
                  return false;
            }
            //return true if the supplied password and the actual password are
equal
            if (passw.equals(password)) return true;
            return false;
      }

      public void doGet(HttpServletRequest request,HttpServletResponse response)
            throws IOException, ServletException
      {
            //get the parameters from the request
            String username=request.getParameter("username");
            String password=request.getParameter("password");

            // get the writer to send html output
            out = response.getWriter();

            // get session object. The parameter 'false' means that if no
            // session object present, do not create one - return null.
            HttpSession session=request.getSession(false);

            // if session is already created and has a 'username' attribute,
            // the user is already logged in
            if (session!=null)
            if (session.getAttribute("username")!=null)
            {
                  //build the page 'AlreadyLoggedIn page'
                  out.println("You already have a session<br>");
                  out.println("<a href='/rewebution/core/index.jsp'> continue
</a>");
                  return;
            }
```

```
            // if the supplied log in information is correcte, then…
            if (validateLogin(username,password))
            {
                  //…build the page "loggedInSuccessfully"
                  session=request.getSession();
                  session.setAttribute("username",username);
                  out.println("you have been logged in. <a
href='/rewebution/core/index.jsp'> continue </a>");
            } else
            {
                  // …build the page BadLoginInformation
                  out.println("wrong user name or password");
                  out.println("<a
href='/rewebution/core/login.jsp?username="+username+"&password="+password+"'>
continue </a>");
            }
      }
}
```

Code example 3 – ValidateLogin servlet

8.6 JavaBeans and Enterprise JavaBeans

Both beans are considered to be a component model. Though the names are almost the same, this is the only thing which is similar.
JavaBeans is intended to be used for intra-process purposes. It can be used to solve a variety of problems, but is primarily used to build clients by assembling visual and non-visual widgets. Original JavaBeans were not meant to be used for distributed components.
Enterprise JavaBeans defines a server-side component model that allows business objects to be distributed to several different machines. The EJB component allows programmers to focus on business logic while EJB server is responsible for making the component a distributed object[1]. The component distribution among different machines is the key difference between JavaBeans and EJB.

8.7 JSP defined

JSP is a type of Java servlet that is designed to fulfill the role of a user interface for a Java web application. JSP are written as text files that combine HTML code, XML elements and includes predefined JSP actions and tags.
Servlet container automatically compiles and converts JSP pages to Java servlets and instantiates to handle requests.
There is quite a number of ways and elements to write into pages:

- **Java code (java servlets and expressions).**
- **JSP expression language elements**
- **JSP tags**.
- **Java Beans**

Before discussing all coding ways individually and presenting some examples from our Rewebution system, we will discuss the Model View Controller architecture of the system.

[1] EJB server – besides making component a distributed object, EJB server has other responsibilities such as managing transaction, persistence, concurrence, and security services.

8.8 Model View Controller architecture

The ways of programming a JSP pages must be carefully chosen depending on the project size, desired scalability, and time frame provided to develop the system (if quality is not the prerequisite).

Java servlets are best when they are used to accomplish complex business logic. This is because servlets have access to almost everything – from HTTP headers and status codes to access databases, generate jpeg images on the fly and perform other tasks flexibly and efficiently. This is true while it not comes to generating HTML output. In fact JSP component is a type of Java servlet that is designed to fulfill the role of a user interface for a Java application. What to do if the web application has to perform business logic efficiently and smoothly, and generate nice to look HTML output. The solution is as simple as possible – use both. Servlets should be allowed to process business logic and then decide which JSP page will create the output. This solution is called the Model View Controller architecture. The bean would be the model[1], view would be JSP, and the controller would be a servlet.

For web applications the process might be summarized in the following simple steps:

- Define beans to represent the data
- Use a servlet to handle the requests
- Populate the beans
- Store bean in the request, session, or servlet context
- Forward the request to a JSP (take a look to the subsequent section discussing the use of forwarding requests versus redirecting).
- Extract the data from the business and generate the output

Though this architecture might sound really good at the first glance, it nevertheless should always be considered and examined carefully before using it. As we have stated before, project size and scalability greatly influence the decision on MVC architecture. The bigger the project and/or development team, the more elaborate techniques should be used. In the following table we present the strategies for invoking dynamic code from JSP.

Simple application or development team ↓ Complex application or large development team	**Call Java code directly**. Place all Java code in JSP page. Good only for very small amounts of code
	Call Java code indirectly. Insert Java code only to call other classes.
	Use Beans. Develop separate class and invoke it through jsp: useBean, jsp: getProperty, jsp: setProperty.
	Use the MVC architecture. Have a servlet respond to original request and store results in beans. Forward to a JSP page to present results. JSP page uses beans.
	Use the JSP expression language to access objects. Usually used in conjunction with MVC.
	Use custom tags and invoke them with XML-like syntax

Table 2 – Strategies for invoking dynamic code from JSP[2]

This table clearly shows what techniques project should use. As with our small project, the eligible strategy would be the MVC approach and using beans. We say MVC approach (instead of architecture) because we did not apply MVC on overall system design. Only some parts of the system (for example use case UC107 Put equipment to cart) require more elaborate design, whilst the other system parts can be implemented straightforwardly only with JSP and minimum of Java Code.

[1] The beans in our project are used more as an easier way to transfer data from one page to another. The model in our project is still considered to be the database.
[2] The table was adapted from the book Core Servlets and Java Server Pages Volume1: Core technologies by Marty Hall and Larry Brown 2nd edition, page 436.

8.9 Implement MVC architecture with forwarding or redirecting

Basically the MVC architecture may be implemented either by forwarding the request from servlet to JSP or by using redirect.
The forwarding is made with a RequestDispatcher object (the <jsp:include> tag is evaluated to this object during JSP translation phase). The object is obtained by simply calling the GetRequestDispatcher method of ServletRequest object. This method is more preferred when the address of the web resource (the URL) needs not to be revealed (that is, the name of the JSP page remains unknown for the user, because the location is not changed). In addition, the control is transferred entirely on the server and no network traffic is used.
Using redirecting technique is beneficial when user should be able to see the JSP page location (most notably to bookmark this page). However, sending a redirect involves an additional round trip between client and server, since the server sends to the client the response with status code 302 – meaning that the destination is being changed.

As we have said, sending a redirect should be used when the JSP page makes sense not only in the context of servlet generated data. Having this in mind, we have implemented component addToCart.jsp like this:

```
<jsp:useBean id="cart" scope="session" class="Cart.Cart" />
<jsp:setProperty name="cart" property="*" />
<%
      // instruct cart bean to process the request by registering all equipment
      cart.processRequest(request);
      // send redirect to show the contents of the cart
      response.sendRedirect("cart.jsp")
%>
```

Code example 4 – send Redirect

Probably the other reason why we implemented addToCart.jsp with redirecting is because we did not use here MVC architecture☹ (we just did not know the architecture it by the time of implementing it).

8.10 Java scriptlets and java expressions

The code inserted into JSP pages directly between the special marks <% and %> is called java scriptlet. The Java code is compiled during translation of JSP page into servlet. The code inserted between <%= and %> is called java expression. The code is evaluated at run time and inserted into the servlet's output (unlike with java scriptlets, where you have to insert output manually into jsp).

JSP has a number of predefined variables (or "implicit objects") to simplify the expressions and java scriptlets. These are:

1. request, the HttpServletRequest
2. response, the HttpServletResponse
3. session, the HttpSession associated with the request
4. out, the Writer used to send output
5. application, the ServletContext. This is a data structure shared by all servlets and JSP pages in the web application.

The following example is an excerpt from the file "standart_panel.jsp". This file generates the panel with links to about, help and home pages. Furthermore, Java code is used to test if the user is logged in and conditionally insert appropriate links either to Login or Logoff pages.

```
<!--create a div section and specify its position using absolute
coordinates -->

<DIV ID="Standard panel" STYLE="position:absolute; top:3px;
left:560px; width:233px; height:52px; z-index:0;">

<!-- create some links -->
     <a href="../core/about.jsp"> About </a>
     <a href="../core/help.jsp"> Help </a>
     <a href="../core/index.jsp"> Home </a>

<!-- if user is logged, print users name and add links to cart and
log off pages -->
     <% if (request.getSession().getAttribute("username")!=null)
     {%>
          <br> user name =  <% out.println(request.getSession().
getAttribute("username"));%>
          <a href="../cart/cart.jsp"> Cart </a>
          <a href ="../core/logoff.jsp"> Log off </a>
     <% }else {%>
          <!-- else add -->
          <a href="../core/login.jsp"> Log in </a>
     <% }%>
</DIV>
```

Code example 5 – conditionally inserting HTML text with Java Code

8.11 JSP expression language (EL) elements

This is a scripting language based on JavaScript. The functions might be basic arithmetic, comparison, accessing implicit objects (like request, session, header, cookies and others) and calling functions defined in java classes. EL was originated by the JSTL, which we will discuss in the next section JSP tags.

We did not use EL elements in our project. However, we could have used them for example in getting parameters from request. Let's recall the situation where the user submits the items he wants to put in the cart. Instead of using bean to get all the data, we could write like this:

```
${sessionScope["Item"]}
```

This simple statement would get the parameter Item.

8.12 JSP tags.

The Java Server Pages Standard Tag Library (JSTL) specification defines the tags that can be used directly in JSP like any well know HTML tag. JSTL tags are based on XML and thus are very easy to get familiar with. JSTL provides very broad functionality from xml processing and data formatting to SQL and database access. Standard helper tags such as setting object attributes or iterating through objects in a list are also included. JSTL also includes a standard action tags which implement the most common functions needed in JSP.

We have used JSP tags throughout the project for including standard panel with "login", "help" and "about" links in every page by writing `<jsp:include page="../core/standart_panel.jsp" flush="false"/>.` This way we ensured that the style contains its standard look-and-feel.
Here is the whole example of the Home page.

```
<html>
      <head>
      <title> Welcome to Roskilde Grej online  </title>
      <!-- include cascading style sheets, defined in file style.jsp -->
      <%@ include file="../core/style.jsp" %>
      </head>
      <body>
      <!—include standart panel -->
      <jsp:include page="../core/standart_panel.jsp" flush="false"/>
      <h1> Welcome to Roskilde Grej on-line </h1>
      <!—add link to equipment list home page -->
      <a href="../equipment_list/equipment_list_home.jsp"> Rent Equipment </a>

      <!—add a count tag to display the number of visits to our site -->
      <%@ taglib uri="/WEB-INF/tlds/count-taglib.tld" prefix="rewebution" %>
            <rewebution:count />
      </body>
</html>
```

Code example 6 – JSP tags in use

As can be seen in example, the use of tags is very simple.

It is also possible for developer to create his own custom tag and even custom tag library. In the previous code example you can see 3 lines

```
<%@ taglib uri="/WEB-INF/tlds/count-taglib.tld" prefix="rewebution" %>
<rewebution:count />
</body>
```

These lines define the custom tag library to be used and actually use the count tag to display the number of requests for this page.
We have chosen to create tag library for the benefits we get. Firstly we clearly separate view and code. Secondly, the created tag is easy to use and even a non Java aware web page designer could use the tag, because of its simple, clear, and non confusing syntax. Without this, java code would have to be inserted directly into the JSP (for example setting and an attribute to ServletContext representing number of visits).

In order to create custom tag one must define it in two steps:

- Firstly a Java **tag handler** class is created to provide tag's functionality. In order to achieve this, a Java class just has to extend one of the Tag Support classes. One of them is called TagSupport class.
  Another solution would be to write a **tag file**. A tag file defines a custom tag in JSP syntax. Later the web container generates a Java class from the tag file automatically. This way tags developer life is made easier.
- Secondly, a tag library descriptor (TLD) must be created to provide a mapping between a tag and the tag handler class. A TLD is configuration file written in XML syntax.

Here is a tag handler example.

```
// create a tag class
public class CounterTag extends TagSupport
{
      // override the doStartMethod define in TagSupport. This method
      // processes the start tag for this instance.
      public int doStartTag()
      {
            try
```

```
            {
                  //  get the ServletContext object used to store our count
variable
                  ServletContext application =pageContext.getServletContext();
                  // get the count attribute from the application context
                  Count count = (Count)application.getAttribute("count");
                  // I f the count object does not exist yet, create it
                  if (count == null)
                  {
                        count = new Count();
                        application.setAttribute("count", count);
                  }
                  // build output
                  JspWriter out = pageContext.getOut();
                  out.println("<BR CLEAR=\"ALL\"><BR><HR>");
                  out.println("This site has received "+count.getCount());
                  count.incr();
            } catch(IOException     ioe)
            {
                  System.out.println("Error in CounterTag: " + ioe);
            }
      // return value to skip the body evaluation for the current tag since it
is a bodyless tag
      return(SKIP_BODY);
      }
}
```

Code example 7 – Count tag Tag Handler class

Here is the example of our TLD file for count tag

```
<!-- define xml version and document type -->
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib>
<!-- start defining tag lib -->
<taglib>
      <!-- define lib version, jsp version and short name for tag -->
      <tlibversion>1.0</tlibversion>
      <jspversion>1.1</jspversion>
      <shortname>Counts</shortname>
      <info>
            A tag library for various counters.
      </info>
      <!-- start defining tag attributes -->
      <tag>
            <name>count</name>
            <!-- defines the handler class to use for tag -->
            <tagclass>counter.CounterTag</tagclass>
            <bodycontent>empty</bodycontent>
            <info>Hit count</info>
      </tag>
</taglib>
```

Code example 8 – Count tag TLD file

8.13JavaBeans

Here we present the JavaBeans.
Generally speaking, a JavaBean class might be created very easily. It is a normal Java class except it has to confine to several simple rules:

- A bean class should have a zero argument constructor
- Should have no public instance variables
- Persistent values should be accessed through setter and getter methods.

We have chosen to use in our project a bean for storing items that were added to cart. Java beans provide three advantages over scriptlets and JSP expression language:

- **No java syntax**
  By using beans, programmer can manipulate date only with the use of simple XML syntax compatible tags. This greatly helps in separating code from view.
- **Simpler Object Sharing**
  The ability to set beans scope value to page, session, or application greatly simplifies sharing the same bean among pages or even whole application.
- **Convenient correspondence** between request parameters and object properties
  Bean component greatly simplifies the process of reading request parameters, converting to appropriate types and putting results to objects.

In Rewebution system we have developed Java Bean named Cart. The beans responsibility is to store the list of equipment user is going to reserve. Bean also provides some data formatting functionality through the method String getItemData(). Here is the shortened outline of the Cart bean class.

```
package Cart;

public class Cart
{
	// create a template class Vector to contain  EquipmentDescriptor objects
	Vector<EquipmentDescriptor> v = new Vector<EquipmentDescriptor>(50);

	// parameters for storing request attributes
	String submit = null;
	Vector<String> item = new Vector<String>(50);

	public void setItem(String[] name)
	{
		for (int i=0; i<name.length; i++)
		{
			item.addElement(name[i]);
		}
	}

	// gets and formats the equipment list
	public String getItemData()
	{
		HTMLEquipmentListDecorator a=new HTMLEquipmentListDecorator();
		return a.decorate(v);
	}

	// private helper class for putting all items into cart
	private void addAllItems()
	{
		System.out.println("item="+item);
		ListIterator i=item.listIterator(0);
		while (i.hasNext())
		{
			addItem((String)i.next());
		}
	}

	// process the request by putting all items into cart
	public void processRequest(HttpServletRequest request)
	{
		if (submit.equals("add")) addAllItems();
		reset();
    }
}
```

Code example 9 – JavaBean class

Having created the bean class we are ready to take the benefits and create the page where all the submitted items are sent and put into the cart. The actual code is really small since all the functionality is provided by the bean.
The bean is accessed with one easy line `<jsp:useBean id="cart" scope="session" class="Cart.Cart" />`. This line tells to use bean with class Cart.Cart and assign to it session scope. This JSP action tag instructs to use the existing bean or create a new one if none exits. The next line `<jsp:setProperty name="cart" property="*" />` instructs to set beans properties to the request parameters. The third sentence calls the method of a bean to actually put the items into the cart and then redirects the page to show the cart contents. Here is the complete example.

```
<jsp:useBean id="cart" scope="session" class="Cart.Cart" />
<jsp:setProperty name="cart" property="*" />
<%
      cart.processRequest(request);
      response.sendRedirect("cart.jsp");
%>
```

Code example 10 – add equipment to cart

The next example of page shows the contents of the Cart. Again the logic is the same – access the java code indirectly through the beans, which was created while adding equipment to cart and display the cart beans contents.

```
<html>
      <head>
      <title> Welcome to Roskilde Grej online  </title>
      <%@ include file="../core/style.jsp" %>
      </head>
      <body>
            <jsp:include page="../core/standart_panel.jsp" flush="false"/>
            <h1> Cart </h1>
            <!-- using bean cart
            <jsp:useBean id="cart" scope="session" class="Cart.Cart" />
            <br> You have the following items in your cart:
            <!-- access bean to get item data
            <%
                  out.println(cart.getItemData());
            %>
      </body>
</html>
```

Code example 11 – show content of the cart

Note that in both examples when using bean a special keyword is used to determine the beans scope (`scope="session"`). The scope might be one of the following:

- Request scope – meaning the bean will be valid only in the current request
- Session – the bean will be valid in the current session (in this case – will be valid as long as the user is logged in)
- Application – meaning the bean will be valid as long as the web-server is running (useful to share global resources, for ex: database connections).

8.14RMI

While the previous examples were concerned mainly with Thin Web Client architecture, we are now to discuss the outcomes of possible use of Web delivery architectural pattern in Rewebution system.

RMI stands for Remote method invocation (as was already stated in section discussing Web Delivery pattern). RMI is the Java product and specifies how two Java components can commu-

nicate over distance (or in different run-time environments). If the communicating components happen to be not Java, but c++, COBOL, ADA or others, using IIOP[1] is the solution.

The difference between RMI and IIOP is that IIOP is more of specification and RMI is a concrete product for Java components. Apart from subtle implementation differences, the target components are the main differences for these two technologies.

During interaction modeling for distributed objects, the details concerning set-up of the distributed components usually are modeled separately, most notably in deployment diagrams. This way the modeling of distributed components happens to be almost the same as for the rest of the system. The only thing added is the stereotype <<RMI>> or <<IIOP>, depending of the technology used.

For the Rewebution System we have chosen Thin Web Client. Let's imagine that we have to design UC107 Put equipment to Cart having distributed components. This decision dramatically changes our architecture. Firstly, the Cart object now is to reside on client-side, since cart holds only client-specific information and there is no keeping this information remote. Displaying the cart contents now includes accessing database on the server-side and acquiring equipments details. This acquiring will be done through the use distributed objects. The cart contents probably would be best to show by developing a JavaBean having a visual representation as a spreadsheet or a list box with equipment name, equipment type, cost, description and other columns.
Here is the possible **class diagram** for this solution

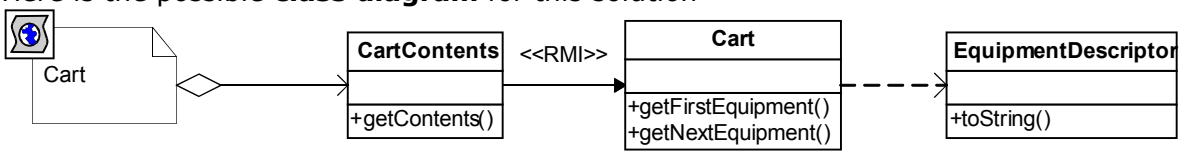


Diagram 16 – UC107 – Class diagram with distributed components

CartContents is a JavaBean component accessing Cart object. Cart component connects to a server-side component capable of returning EquipmentDescriptor classes.
The following diagram is a possible **sequence diagram** for this solution.

[1] IIOP: Internet Inter-ORB Protocol is OMG's CORBA protocol for interacting with distributed objects across the Internet or any TCP/IP based network.

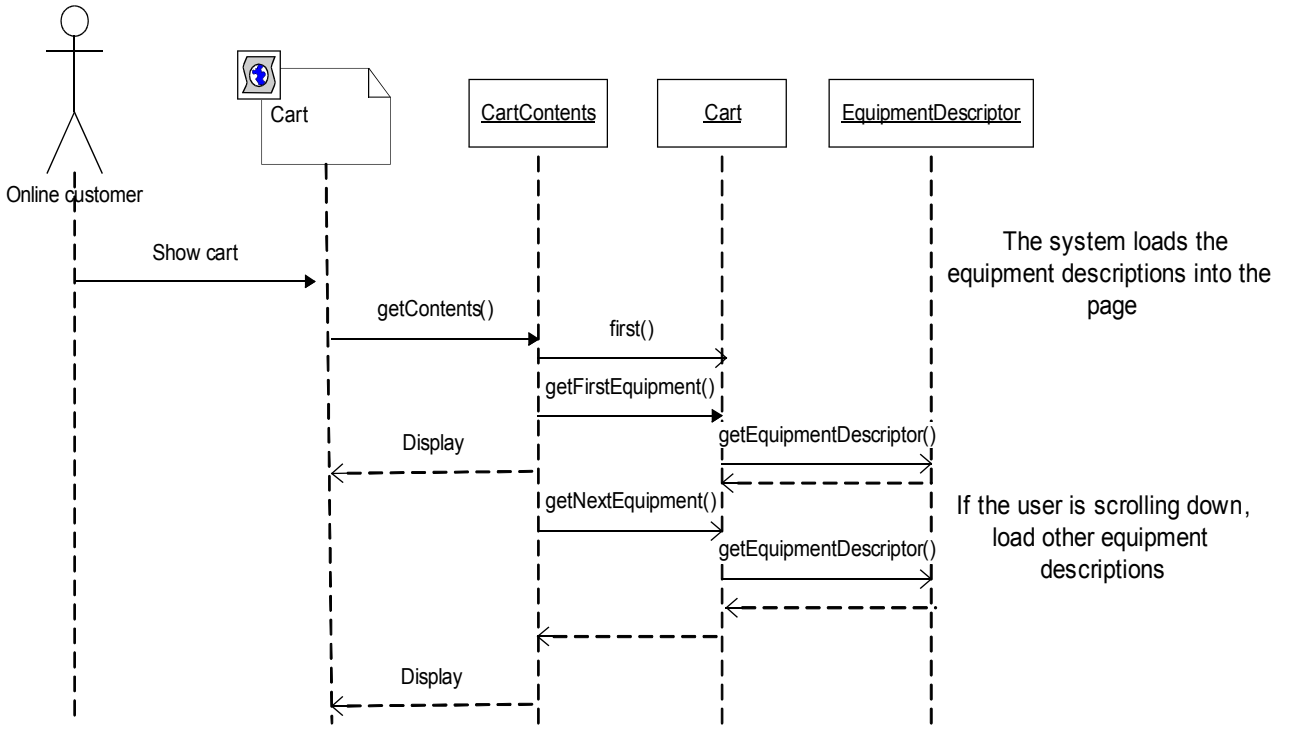


Diagram 17 - UC107 –Sequence diagram with distributed components

The underlying client-server communication between server and client-side components is not shown in the sequence and class diagrams. This allows to model necessary system parts effortlessly leaving out the details for later inspection. The following diagram visually illustrates the common technique for implementing distributed component communication with the RMI.

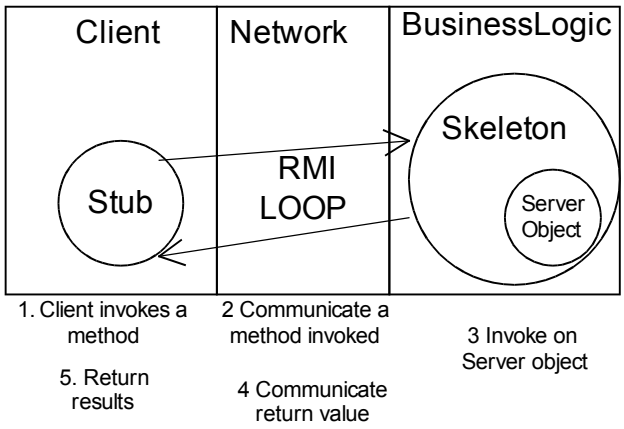


Diagram 18 – RMI loop [1]

Having this diagram it is clear that we will have to create a stub and a skeleton. The stub will act as a proxy for the actual object, existing on the server. The client-side objects will interact with the proxy, the proxy in turn communicates with skeleton and skeleton executes the requested method. The results are sent back to the client.

8.15Sessions

One more important question for the web application development is how to track user activity. While with the Web delivery the solution is quite simple and straightforward, with thick and

[1] The diagram was adapted from "Enterprise JavaBeans" 3rd edition by Richard Monson-Haefel. Published by O'Reilly 2001

thin web client it is not so trivial. The reason this difference exists lies in the type of protocol used for communicating between server and client. The web delivery system does not use so called stateless (or connectionless) HTTP protocol. The decision to use HTTP protocol implies that no persistent connection will be made between the client and the server thus providing no means to track user's state. Other protocols establish a permanent connection from the client to the server and basically this connection can be utilized as user-state monitor.
However, the problem with connectionless HTTP protocol is solved by introducing sessions. A session represents an interaction between a web user and a web application. As in our Rewebution System, the web user must first log into the system, search and rent some equipment and log off the system. All these user's actions would be considered completely separate request from the view of the server (which uses HTTP connectionless and stateless protocol). These sequential visits by a single user to our system are considered part of one session. Basically there is two ways of implementing sessions – cookies and URL rewriting. (A cookie is name/value pair of information on a user's computer that a web).Both of the methods use the session ID – a special value, usually a long string representing numeric value in hexadecimal notation. This value is made available to the web application either by:

- Storing cookies on client web browser containing the session id.
- Or Rewriting URL so the request URL contains the session id value.

The URL rewriting method is used mainly when the cookies are disabled by the browser.

Talking specifically about Java web programming, the web page author has the access to HttpSession object, which represents a session. We have used the session object implementing the ValidateLogin component (logs in the user). The code example was already described in Java servlet defined section. Here we present the code example showing how the user can be logged off by simply invalidating the session object. The code can be put into servlet or as a scriptlet in JSP.

```
HttpSession s=request.getSession(false);
s.invalidate();
```

Code example 12 – invalidating session

8.16The HttpSession and ServletContext objects application

The HttpSession object can be used not only to track the user's activity on the web site, but also to store specific information for the user such as shopping cart items.
In our Rewebution system, we have used the JavaBeans to represent the cart and store equipment. We could have taken the other approach – that is instead of using JavaBeans, use a normal class and associate that class with client session.

In Java web applications one more architecturally significant object resides – that is the ServletContext object, more commonly referred to simply as "the application". This application object similarly to the HttpSession object allows storing user information. However, these two objects significantly differ by scope. The session object keeps the specific information for the specific user, while the application object holds common information for all web application. This means that two different web users having two different sessions are able to access the same application data.

8.17Database connections and connection pooling

Even slightly elaborated web application employs databases. The layer for databases(DB) is called generally a Persistence layer. This is because persistence layer hides the details of the actual method to achieve persistence used – should it be databases, files or other.
As for our Rewebution system, we implemented persistence layer by using a simple java class from the previous semester project – the Rental System. This class (named DBManager) opens statements, executes the queries and responsible for connecting to the database. The class will

not be discussed here, since it was already parts of previous project report. However, we will discuss the topic connection pooling.
The connection pooling techniques undertakes the problem of solving the costly creation-deletion problem of database connections. This kind of problem arises when the web application has different simultaneous users accessing database (and of course it will happen probably most of the time). The essence of the problem is that many users will create/delete many connections at the same time and it will consume lots of server resources. To reduce server's resource usage DB connections can be shared among servlets and JSPs.

8.17.1Hand made approach

One way to implement connection pooling is to code the class specially addressing this problem. The class could be made a singleton class (meaning it can have at most one instance of that class) and be shared among the servlets and JSPs by associating it with the ServletContext object. This way the web page author could obtain reference to the connection pool class by simply calling getAttribute() method of the ServletContext object.
While this solution might be OK for small projects like Rewebution system, more elaborate and industry accepted solutions should be used.

8.17.2Using JNDI and DataSource

JNDI stands for Java Naming and Directory Interface. In this section we discuss how to access a database resource by using JNDI.
JNDI is the most efficient[1] (and probably the most common) method of accessing database resources in portable manner. It is a Java API that is designed to store objects in a hierarchical tree structure. Servlets and JSPs can use methods of the JNDI API to obtain references to resources.
For database code, this implies obtaining references to javax.sql.DataSource objects, which are factories for database connections. The DataSources provide connection pools. The pool manages a specific number of database connections and tells the connection pool users which connections are free to use and which are occupied. This way a number of connections are reused and efficiency increased. The following code examples illustrate how JNDI and DataSources could have been used to implement connection pooling.
The first example shows the initialization of JNDI and obtaining reference to the DataSource object.

```
Javax.naming.Context  env=null;
try
{
      Env=(javax.naming.Context) new javax.namin.InitialContext().lookup
      ("jva:comp/env");
      // look up a data source, which represents a connection pool
      Javax.sql.DataSource pool=
                  (javax.sql.DataSource) env.lookup("jdbc/rewebution");
      If (pool==null)
      throw new ServletException ("jdbc/Rewebution is an unknown DataSource);

}
Catch (Exception)
{
      …
}
```

The second example shows how to get a connection and execute a simple statement.

```
// get a connection from the connection pool
Java.sql.Connection con=pool.getConnection();
// create a statement
String stmt=con.createStatement();
// execute a query
Java.sql.ResultSet result=stmt.executeQuery("select name,cpr from customer where
login='a' and password='xxxx'");
```

[1] Taken from "Java servlet and JSP cookbook" by Bruce W.Perry, published by Oreilly 2001.

8.17.3Rewebution approach

In our project, unfortunately, only one connection is used in whole application. It is the approach we do not recommend anybody to use ☺. The same single connection is used to create statements and execute queries. However, we did not find that only a single statement can be used at one time for the single connection ☺. It surely works for one web-user at a time, but no guarantees are given for other cases. Surely we would implement a connection pooling or at least create a new connection for every new user, but despite the limited time frame, we have not done it.

9 Conclusion

This is the last section of the project – the evaluation.
Though the project is finished, we have learned a lot of new things and are able to define our own project's flaws. The project satisfies for the goals we wanted to achieve in the beginning of the project. Yet now we are not really satisfied with the results. It is the humanity's essential thing which makes evolution possible – the dissatisfactions with current situation and perpetual strive for perfection.
The project is strong in the variety of topics discussed and in the abstract point of view used to describe them. However, the project lacks the skills in literary language as well as understanding reader's point of view. Did we explain the topics enough clear to be able to understand them even for readers who are not very much familiar? Probably we did not explain and relate our code with the theory discussed.
The secondary goal of the project was to make this report conveying knowledge to the reader. The work is to be valuable only then, when the reader is gaining some tangible or intangible benefit. It is also necessary for the work to influence readers mind and inspire some discussions and thoughts, i.e. to be the pabulum (the food) for the reader. If the reader's mind is left untouched, the work is useless ☺ (at least to some extent).

With these words we finish the report. ☺ Bye bye.

10Literature

- [Chaffey 2004]
  "E-business and e-commerce management" by Dave Chaffey 2004. 2nd edition.
- [Web-RUP 1999]
  "Building Web Solutions with the Rational Unified process: Unifying the Creative Design process and the Software Engineering Process", a Rational Software and Context Integration white paper.
- [WPerry2004]
  "Java Servlet and JSP cookbook" by Bruce W.Perry. Published by O'Reilly 2004. 1st edition.
- [Haefel]
  "Enterprise JavaBeans" by Richard Monson-Haefel. Published by O'Reilly 2001. 3rd edition.
- [Nielsen]
  http://www.useit.com/papers/heuristic/heuristic_list.html
-]McConnell]
  http://www.amazon.com/exec/obidos/ASIN/1556159005
- [Avisson]
  "Information systems development methodologies, Techniques and Tools" by David Avisson.

11Appendices

11.1Appendix A - "XP versus UP – a methodology comparison study work"

XP versus UP

A methodology comparison study work

Hamlet's Castle (Kranberborg)
Early spring 2004, ©

Darius Damalakas
November 5, 2004

Roskilde Business College, 2004
3rd semester, Advanced Computer Science

Introduction

The purpose of this document is to analyze the two methodologies XP and UP using the framework proposed in "Information Systems development" [9]. The analysis is carried out for the studying purposes and thus the reader should be aware that the analysis will not be of the same depth in all aspects and may contain errors or inconsistencies. The author would be very thankful for comments and advices (e-mail: darius.damalakas@g-mail.com).

The comparison will be made based on information from official XP web site [1] and practice gain from following the approaches described in Craig's Larman book [10] "Applying UML and patterns" and 'The New Methodology" [11] from Martin Flower.

Table of Contents

1Introduction.....3
2Project Start.....6
3Problem Analysis 7
4Project Scope.....8
5Project start.....9
6User participation and user interface design.....11
7Client-Server architecture 19
8Web programming 31
9Conclusion.....48
10Literature.....49
11Appendices.....50

Philosophy

These sections present the philosophy the methodologies expose. In this context, 'philosophy' is regarded the same as in the book "Information Systems development" [9] it is a principle or a set of principles that underlie the methodology.

Paradigm

Two paradigms are considered relevant to this are the science [6] and the systems [7] paradigm.

It is difficult to strictly say to what paradigm methodologies should be put. We identify UP as being science paradigm while XP being systems paradigm. This partitioning is on the biggest part influenced by the overall perception of systems development for particular methodology. XP tries to build the system gradually. On the other hand, we believe, that UP is based on science paradigm because it tries more or less to build the system which is based on elaborate architecture (because UP is architecture-centric). Of course UP (as also XP) is based on an iterative approach.
The important issue is not the exact paradigm we assign, but the discussion our ideas might generate.

Objectives

This section distinguishes the objectives of the methodologies, or more strictly – it tries to answer the question 'could the use of methodology lead to the implementation of a purely organizational or non-IT solution?'

The objective of the XP methodology is not to come up with the non-IT or purely organizational solution, but quite the opposite - XP is designed to build IT-systems and do not address the analysis of the organization strategy and functions. UP is identified also as capable of producing only IT solution. This is based on the fact that both methodologies do not incorporate business analysis steps.

Domain

This section describes methodologies as being of the planning, organization, and strategy type or problem solving methodologies. This is related to the objectives of the methodology, but focuses on what aspects or domain the methodology seeks to address.

The primary focus is to solve a particular problem (to create a system which helps achieve some specific goals). That means the domain of interest is rather narrow and solely limits to creating an IT-solution and non-IT solution is not in the area of interest. XP does not address the general planning, organizational, and strategy of information and systems in the organization analysis. UP is also described as problem-solving methodology for the same reasons.

Target

This section is concerned with the target system to be developed, or more particularly whether the methodology is aimed at particular types of application, types of domain, size of system, environment and so on.

The XP is suited, as stated in XP web site [1], for small groups of people, usually varying between 2 and 12, though larger groups of 30 have also report success. It is not possible to use XP with huge staff.
XP enables to embrace change, so projects with highly unstable project requirements fits perfectly.

UP is considered to be general purpose methodology, although it is suggested they are not really helpful for simple, limited systems.

Model

This sections concerns with the model or models that the methodologies use. This can be described in terms of the type of the model, the levels of abstraction of the model and the orientation or focus of the model.

XP uses OO modeling techniques. It is stated that there is some written documentation for the software, more commonly interpreting requirements than documenting design.

In UP the basic models are also of object-orientation approach. However, the models are oriented not only for

software design, but also business processes.

Techniques and tools

This section describes techniques and tools used by the methodologies

UP techniques

UP utilizes the following techniques in developing IT-solution

- Entity modeling
- Gannt charts
- OO techniques and UML (class diagram, use case diagram, interaction diagram, sequence and state chart diagrams, activity diagram)

UP tools

UP does not specifically address the tools to be used together with methodology.
Some individual tools that might be used with UP

- For diagramming:
  Microsoft Visio, MagicDraw.
- For project management
  Microsoft project
- Requirements:
  Microsoft office, Open Office, any XML technology supporting program

- Application packages (supports whole process)
  Rational Rose

XP techniques

XP does not specify techniques to be used except that it is object oriented methodology and appropriate tools and techniques should be used. Based on this we can state that it uses

- OO techniques and UML (class diagram, use case diagram, interaction diagram, sequence and state chart diagrams, activity diagram)
- Requirements are modeled with the use of CRC cards. CRC stands for Class, Responsibilities, and Collaboration. The cards are rarely filled up in great detail and in most cases only a class name is written at the top of the card.
- User stories [3]

XP tools

XP does not specify any specific tools to use except the very important area – testing.

- Unit Test Framework

Unit test framework is a development tool to facilitate creating and managing tests. “Your unit test framework can help you formalize requirements, clarify architecture, write code, debug code, integrate code, release, optimize, and of course test.”

Scope

Both methodologies are valued against the scope of software development life cycle stages they cover. This kind of comparison has major flaw – not all methodologies do follow a life cycle and follow other approaches. As is with our case, both methodologies use iterative approach.
The values for evaluating stage coverage ranges from 0 (means methodology does not cover the stage at all) to 3 (covers in detail). The comparison is based on our quick investigation so the table should not be understood as concise and in most cases correct☺.

Phase/ Methodology	XP	UP
Strategy	0	1
Feasibility	0	1
Analysis	1	3
Logical Design	2	3

Physical Design	1	3
Programming	3	1
Testing	3	2
Implementation	1	3
Evaluation	0	2
Maintenance	0	0

The comparison clearly shows that the UP covers most of the areas rather in detail while XP focuses on programming and testing. This is because XP is a lightweight methodology [2]. XP focuses on delivering the software when it is needed and embraces changes gently through short-time boxed iterations (early feedback) and simplicity.

UP starts with inception phase looking at whether the project is feasible, and proceeds further with the project requirement gathering and analysis. XP does not have an implicit analysis phase – customer is available all the time and writes user stories [3]. User stories serve as a replacement of a big requirements document.

XP explicitly does not state logical or physical design stages – though spike solutions are created to help resolve various technical or system design problems. UP, on the contrary, has a design discipline, where design and data models are created with the Software Architecture document.

XP covers programming stage thoroughly. It has many, though simple rules and practices to guide the programmer of achieving simple code with extendable design. The coding rules are as follows: the customer is always available, code must be written to agreed standards, code the unit test first, all code is pair programmed, only one pair integrates code at a time, integrate often, use collective code ownership, leave optimization till last, no overtime working (instead change the project scope or timing). UP explicitly assumes that the OO programming language will be used (as is the methodology itself OO).

The last stage XP covers thoroughly is testing. Testing is one of the core roots of XP. It is even said “We would like to have more lines of test than we do of actual code” [5]. Both acceptance tests and unit tests are used. Unit test should be in most cases 100% automated. UP pays a great deal of attention to testing also, though not so “aggressively” as XP. In a testing discipline a test model is created with lots of test cases, and, presumably, automated tests.

Outputs

Introduction

The next sections will be investigations of what is actually produced in terms of deliverables at the end of each stage of methodology.

XP Outputs

- User Stories [3]
- CRC cards
- Release plans
- Iteration plan
- Unit tests
- Acceptance tests

UP Outputs

The outputs for UP will be categorized by the UP disciplines. Only the major artifacts (deliverables) will be named here and many simple non major artifacts will be skipped.

- Business modeling
  - Domain Model
- Requirements
  - Use-Case model
  - Vision
  - Supplementary Specification
  - Glossary
- Design

    - Design Model (Use Case Realizations)
    - Software architecture document
    - Data Model
- Implementation
    - Implementation model
- Project Management
    - Software Development plan (includes Project plan, iteration plans, etc.)
- Testing
    - Test Model
- Environment
    - Development Case

Practice

Background

The background of the methodology broadly identifies its origins in terms of academic or commercial.

The UP has mainly an academic background, while XP is a commercial methodology. The new ways of developing software developed by Kent Beck were first tried in DaimlerChrysler and resulted as an XP methodology.

User Base

This topic describes how wide the methodology is used.

Not discussed in this document due to the lack of information

Participants

In this topic we discuss the participants involved, in some contexts these are referred to as 'actors' or 'stakeholders'. The questions answered here will be 'Who is supposed to use the methodology', 'What roles do they perform'. Skill levels are also addressed.

UP participants

A specialist team of professional systems analysts and designers perform the analysis and design aspects and professional programmers design the programs and write the code. The system is implemented by the analysts. In order to learn UP, significant training and experience is needed.

XP participants

Participants are "ordinary programmers", who do not need a Ph.D. to use XP. Part of the team are also managers and customers as well.
Through XP being a lightweight methodology [2], no intensive and expensive training is needed to wield the methodology.

Product

This section describes what is supplied when purchasing a methodology and at what cost.

UP products

The author of this document was not been able to find a freely available documentation.
Software to support the methodology (in activities such as configuration and change control, design and analysis) is advisable, though suitable open source software might be downloaded freely from web.

XP products

The descriptive documentation of the methodology is available online and is presented to the public with no charge. XP does not require any additional software to use it.

Conclusion

The two compared methodologies XP and UP are quite different from their background.

XP is designed to be a lightweight [2] and agile methodology. It is intended to be not used in projects with huge staff (suggested team size varies from 2 to 12 people). XP focuses on following practices which makes software development go faster and remove practices which make it move slower – it means documentation is made only when it is needed.

UP is quite the opposite – it is designed for big projects and with huge staff. A thorough process must be followed in order use UP methodology.

Both methodologies embrace change, though there are some differences in the two: UP embraces change through the use iterations and planning. XP uses also an iterative approach though with much smaller iterations (one of the reasons is because of the huge staff and the communication taking place there) thus enabling considerably fast feedbacks.

It is clear that the differences are primarily implied by the size of the staff. This allows XP to reduce documentation to the minimum and still keep the communication active and efficient.

Another underlying difference between these two methodologies is the way they treat system design. UP is based on an idea that as soon as the design is stabilized, coding planning and programming can take place. While this is a very good idea in civil engineering (where the cost of designing a bridge is 10% of the whole cost of the project), McConnell (http://www.amazon.com/exec/obidos/ASIN/1556159005) suggests that for a large project, only 15% of the project is code and unit test, an almost perfect reversal of the bridge building ratios. He states that the design part takes not less than 50% of the work. XP addresses this (and all agile methodologies [8]) by focusing on coding thus limiting design (and documentation) activities.

UP is

- architecture centric
- Use-case driven
- Iterative
- Deals high risk issues in first place
- Object oriented

XP emphasizes:

- Team work
- Customer satisfaction by delivering software which is needed when it is needed
- Improvement of software project in four essential ways: communication, simplicity, feedback and courage
- Writing tests before actual code to be tested
- Implementing things you need right now, not the things you will need.
- Do documentation whenever it is needed
- The quality of the source code
- Small iterations
- Customer involvement

Whilst there are many differences, the authors personal opinion is of that these two methodologies do not compete with each other – both stresses different parts of IS development and thus provide different approaches and practices.

References

[1] XP website

www.extremeprogramming.org

[2] Lightweight methodologies

An excerpt from XP website [1]: “A software methodology is a set of rules and practices used to create computer programs. A heavyweight methodology has many rules, practices, and documents. It requires discipline and time to follow correctly. A **lightweight methodology** has only a few rules and practices or ones which are easy to follow”.

[3] User stories

An excerpt from XP website [1]: "User stories serve the same purpose as use cases but are not the same. They are used to create time estimates <…>. They are also used instead of a large requirements document. User stories are written by the customers as things that the system needs to do for them. <…> They are in the format of about three sentences of text written by the customer in the customers terminology without techno-syntax".

[4] Spike solutions

An excerpt from XP website [1]: "Create spike solutions to figure out answers to tough technical or design problems<…> Most spikes are not good enough to keep, so expect to throw it away. The goals are reduce the risk of a technical problem or increase the reliability of a user story's estimate".

[5] The four project values

http://www.xprogramming.com/Practices/PracValues.html

[6] Science paradigm

Checkland (1981) describes science paradigm as consisting of reductionism, repeatability, and refutation: "we may reduce the complexity of the variety of the real world in experiments whose results are validated by their repeatability, and we may build knowledge by the refutation of hypothesis".

[7] Systems paradigm

Checkland (1981) argues that human activity systems are systems which do not display characteristics of breaking down a problem into smaller parts does not break the whole system. Human activity systems has quite the opposite characteristics – emergent properties (i.e. the whole is greater than the sum of the parts) and perform differently as a whole. The systems paradigm concerns itself for the whole picture, the emergent properties, and interrelationships between parts of the whole.

[8] Agile methodology

Some time ago lightweight methodologies [2] were the term to be used instead of agile, whilst now these terms are used interchangeably. Agile methods "attempt a useful compromise between no process and too much process, providing just enough process to gain a reasonable payoff", an excerpt from Martin Flower document "The new Methodology" [11].
"Agile methods are adaptive rather than predictive, Agile methods are people-oriented rather than process-oriented".

[9] "Information Systems Development"

[10] "Applying UML and patterns".

An introduction to Object-Oriented Analysis and Design and the Unified Process.
Craig Larman, 2nd edition © 2002

[11] The New Methodology

Martin Fowler. http://www.martinfowler.com