Department of Computer Science and Software Engineering
Bachelor thesis in Computer Science, spring 2002
Andreas Oskarsson, Martin Kling, Tobias Norberg

Web Application Development

-a study on UML Web Application Extension

Examiner: Guohua Bai

Supervisor: Ulrika Sjöström


Department of Computer Science and Software Engineering
Bachelor thesis in Computer Science, spring 2002
Andreas Oskarsson, Martin Kling, Tobias Norberg

Abstract

The complexity of Web sites are increasing and transforming into Web applications that contain business logic, interactivity, transaction handling and states. This phenomenon forces the Web developers to adapt more traditional software engineering techniques to keep the Web applications error free, maintainable, reusable, well documented etc.

Many Web developers do not use any engineering techniques at all and design mainly to create as fashionable applications as possible with no regards on the application's functionality. This results in applications that are hard to maintain and with poor functionality.

The purpose with this thesis was to see if the use of a more traditional software engineering technique, namely the Unified Modeling Language with the newly added Web Application Extension, resulted in a Web application with good design regarding the maintainability of the application.

To investigate the maintainability of an application, the maintainability was further divided into three sub criteria: extensibility, reusability and documentation. These three criteria were then applied on a case study were a Web application was designed. From the analysis of the final design, using the three criteria, the maintainability was derived.

The result of the entire investigation showed that the UML WAE had a good support for extensibility, fair support for reusability and very good support for documentation. From these results the main conclusion was derived, that the use of UML WAE resulted in good design regarding the maintainability.

However, the result is limited to our case study and the design created in that case. The result may have been different if the three criteria had been applied on a different case. Another aspect to consider is that the quality of a design is often dependent on the knowledge of the persons that carry out the design.

Acknowledgement

Special thanks to our supervisor Ulrika Sjöström, Department of Software Engineering and Computer Science, at the Blekinge Institute of Technology. For the guidance and help in the production of this thesis.

For insightful feedback, on our criteria and metrics, we thank Miroslaw Staron, Ph.D. student at the Department of Software Engineering and Computer Science, Blekinge Institute of Technology.

Table of Contents

1 Introduction

Following is the result of our Bachelor Thesis in Computer Science at the Blekinge Institute of Technology written in the spring of 2002. Conducted by Andreas Oskarsson, Tobias Norberg and Martin Kling from the class of 1999 in Informationsystem.

This thesis targets an audience that work with Web application development. It is also directed to an audience with interest in modeling or Web techniques.

The reason we chose Web application development as our topic was our curiosity about the fact that none of us had ever heard of a special model language for Web application development. There are several models and methods available for Object-Oriented development, such as the Unified Modeling Language (UML) and the Object Modeling Technique (OMT), but what about models for Web application development?

1.1 Background

To find out more about what models Web developers use, or if they even use one, we conducted a minor preliminary investigation in the initial phase of our work.

We emailed and asked 24 companies that work with Web development and we got answers from ten of them. The answers varied, some of them used their own models that they had created themselves, some used models like Windows DNA, Model View or even brainstorming, but the majority did not use a model.

Why is that? It seemed to be a common opinion that there is not a need for modeling when they are *only* developing Web applications. Someone said that they just throw in the necessary components and *make it work*. Modeling was considered overkill.

This feedback from the Web developers did not match with our belief that modeling before programming software is important. Therefore, we will focus this thesis on an examination of the Unified Modeling Language (UML) and to see if it

is adaptable and usable for Web application development, and if the final result using UML will be a good designed Web application.

1.2 Delimitations

This thesis is delimited to the analysis and design phases, with no attempt to approach the realization or implementation phase, in a Web application development process. We have analyzed and designed a Web application using UML with an extension that is called Web Application Extension, from now on simply referred to as WAE. It is further restricted to the variables we have chosen to study.

1.3 Hypothesis

When developing Web applications the use of modeling languages is not obvious. But we believe that modeling would improve the applications design and it will therefore gain certain advantages. With this believe we have used UML when developing a Web application and then analyzed the result to see if it resulted in good design regarding to maintainability.

Our hypothesis is as follows:

The use of UML for Web application development will result in good design, regarding to maintainability.

In our attempt to resolve this hypothesis we focused on the Web Application Extension that exists to the modeling language UML. We have further divided the term maintainability into three criteria: Extensibility, Reusability and Documentation.

To help draw conclusions from our hypothesis we will answer the following questions:

In what way does UML WAE support Extensibility?
In what way does UML WAE support Reusability?
In what way does UML WAE support Documentation?

1.4 UML WAE

We will use UML WAE to model in our case study. WAE, an extension to UML, was developed by Jim Conallen; the Web Modeling Evangelist at Rational Software Corporation. The extension is specified in the book *Building Web Applications With UML*. [Conallen]

We will use the UML WAE to analyze and design a Web shop that sells products online. The Web shop will contain the following requirements:

- Show products
- Shopping cart
- Checkout point
- Administration functions
- Order handling

For a more detailed specification of the requirements, see 4.1.4 Case Study.

When we model with UML WAE we will use some of the model elements from the standard UML notation in addition to those model elements that the Web Application Extension offers. Following is a brief presentation on what model elements we use and what we mean when we mention the following expressions.

Standard model elements

A *Class* is an element that is used in the standard UML but not in the extension. We will not use class elements when we model but we may mention them in the text. Instead of the class element we will use the elements available from the extension.

A *Package* is a way to organize smaller related elements (such as classes) into groups to increase the comprehension and to get a better view of the system. [Larman]

Extension model elements

A *Page* is the extensions equivalent to a class in the standard notation. The Page element is a Web page that can be in the form of either a server page or a client page. The differences between these are that the client page is a Web page that the client can view but a server page only resides on the server. [Conallen]

1.5 Maintainability

Maintainability aims to make a Web application easy to maintain. The system should require, when it is implemented and in use, only minimal administration. However when maintenance is required, it should be easy to carry out and take a minimal amount of time. High maintainability can be achieved by a good original design and architecture. [Booch]

Some examples of maintenance could for instance be how easy it is to transport the system from one server to another, insert new functions and fix bugs without problems and minimal alterations to the system.

In this thesis we focus on obtaining *good design* and we have specified this as a high degree of maintainability. With maintainability we mean extensibility, reusability and documentation. [Gillies] When we searched for qualities to describe maintainability we came across several other qualities than those we chose but extensibility, reusability and documentation were those that reoccurred and were most emphasized.

1.5.1 Extensibility

The environment that an application works in is always evolving and demands new functions and added capabilities from the application. To manage these changes the design of the application must embrace extensibility. Extensibility means that new capabilities, functions and modifications can be inserted in the application without the need to modify in numerous places in the application. [Kafura]

To measure the extensibility of the application the qualities coupling, cohesion and generalization/specialization have been used.

Coupling

Coupling is a measure of how strong an association is between two pages or two packages. A strongly coupled page relies on many other pages to perform tasks and changes to one page may force changes in the all the interrelated pages. Loosely coupled pages are preferable when designing for extensibility because they do not rely on other pages and can more easily be extended without numerous changes in other pages. [Larman]

Cohesion

Cohesion is a measure of how strongly the responsibility of a page or a package is related and how the elements work together to complete tasks. A package with low cohesion perform many unrelated tasks or perform too many, whereas a package with high cohesion has strongly related responsibilities and does not perform too many tasks. High cohesion in the pages and packages is preferable to keep the maintainability and complexity comprehendible and to support extensibility. [Larman] [Booch]

Generalization/specialization

Generalization is to find common properties among the software's concepts and try to organize them so that the software can work in the most general way possible. One way to generalize is to transform the general concepts into supertypes. From these supertypes, more specialized concepts (subtypes) can be created and related. This activity supports extensibility by creating a hierarchy that can easily be extended by new subtypes. [Larman] [Kafura]

In Web application development the ways to create these sorts of generalization/specialization hierarchies are limited by the techniques that are used to implement the application. However, most of today's existing server script languages support the creation and use of these hierarchies, in the form of inheritance, but the stateless environment on the Web limits them. [Ratschiller]

Coupling, cohesion and generalization/specialization are evaluative qualities and should be applied while considering all design decisions. For more information on design patterns see Larman. These principals are, however, those we found to be most supporting of our demands for extensibility.

1.5.2 Reusability

Reusability is simply the ability of software artifacts (specifications, designs, or source code) to be used again. Software reuse does not just mean recycling the code itself into a new piece of software. In general, it can include any asset, such as specifications, design models, and user documentation, whatever is necessary when implementing or updating an application. [Poulin]

Reuse of existing tested and proven software components correctly can save both time and money when developing new products. Reusing also decreases the number of bugs and the demand of maintenance as that has already been done. Designing systems with a high degree of reusability demands a design that allows adaptation, because the developers cannot predict all scenarios in which the software component will be reused. [Fournier]

Reuse can be divided into white-box and black-box reuse. The white-box strategy is when a developer copies and modifies code from earlier implemented software. This approach has according to Poulin limited benefits, because “the modified component must undergo the same testing, configuration management, maintenance, documentation, and all the other requirements” as if you had newly programmed it. This means that white-box reuse only saves some time in the development phase but not in the maintenance. [Poulin]

Black-box reuse is a more systematic kind of reuse because it must be more planned and more thoroughly worked through. In black-box reuse no alterations to the inserted component are allowed. If a modification is needed then it can be made through inheritance and polymorphism but not in the reused code. Black-box reuse saves a lot of effort in the development phase but also in maintenance. [Poulin]

1.5.3 Documentation

Vice President of Rational Strategic Services Walker Royce states that “design without documentation is not design.” [Royce] Why? Because the models are the future documentation and they must be treated as valuables. Documentation seeks to visualize analysis and design decisions to the maintainers of the system. [Booch]

Documentation is essential but should not drive the process. Elements to be documented are:

- End-user documentation (not covered in this thesis)
- Analysis models
- Architectural visualizations
- Implementation documentation (not covered in this thesis)

The documentation can then be used when altering and updating the system. Booch points out that documentation should be on a high-level basis not containing semantics of each method on a “class-by-class” basis. [Booch] Another essential quality to documentation is that it should have good traceability.

Traceability

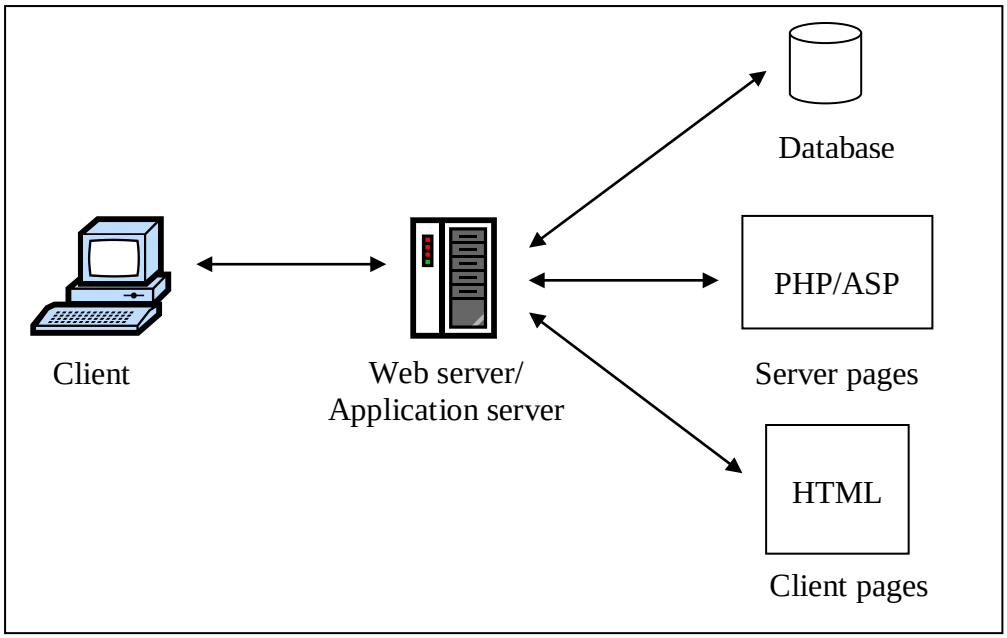
Good traceability refers to the ability to easily follow and describe the initial requirement to the final implemented code, in both a forward and a backward direction. Good traceability should answer the question: In which parts of this program is functionality X implemented? [Gotel]

2 Web applications

When we are discussing Web applications we do not mean a static Web site like a personal homepage. Chief scientist Grady Booch on Rational Software Corporation describes a Web application as “much more dynamic, full of rich content and capable of changing the state of the business as a result of user interaction”. [Conallen]

A Web application is a site that has invoked business logic, interactivity, transaction handling and states. [Ratschiller] [Conallen] The three components to achieve this are a browser, a Web server, and an application server. Often a database server is added to make the application more dynamic, see picture 2.1. [Conallen]

A Web site does not ordinarily involve issues such as security and usability factors. But Web application is a security risk, crackers can redirect your traffic and take your clients credit card numbers and the like, so methods to secure the application must be applied. [Ratschiller]



Picture 2.1 - Web application architecture

2.1 Web applications vs. Client/Server applications

Client/server describes the relationship between two computer programs, one on the client's computer and the other one on the server, in which the clients program makes a request to the server's program that fulfills the request. [Sullivan]

A Web application basically functions the same way, but with a browser and a Web server. Instead of using an installed and licensed client program, Web applications use a standard Web browser to connect to the server. [Greene]

In this report we define a Web application by the definition by Jim Conallen as a Client/Server ~~software~~ software system that has, at a minimum, a browser, a Web server, an application server and possibly also a database server.

There are however differences between a Web application and a Client/Server application. GUI, structure, navigation, protocols, speed, security, techniques etc. are issues that can differ, but although the overall differences is rather indistinct. The applications have the same architecture, the functionality is basically the same and they are used in many of the same situations. [Sullivan] [Conallen]

53 System development

In the early 1990's analysis was a small part of the traditional (non-Web) system development process and consisted of about 25 percent of a projects total time. [Hernbäck] Ten years later the analyze part had grown to about 50 percent which clearly shows the development within this subject over the years.

The traditional Web page has been developed to be as fashionable as possible and this approach has caused problems because of the lack of functionality analyzing. Another aspect is that reuse and site maintenance are not considered, leading to difficulties in making modifications but also takes away the possibility to reuse parts of the application, saving time and money in the development process. [Powell]

One earlier approach in Web application development was the Rapid Application Development (RAD) process. The idea behind RAD was a prototype process where you implement an application and revise it in several iterations until it has passed the buyer's evaluation. This approach often created poorly designed or impossible to maintain pages. [Powell]

There are goals that developers should try to fulfill with a page. It should be correct and error free, maintainable, reusable, robust and reliable, well documented etc. [Eklund] [Powell] To achieve this you should use software engineering techniques because Web sites are becoming more and more like traditional software.

Planning should take about 50% of a Web application development project's time [Ratschiller] and aims to prevent spending time with programming on something that would have needed more analyzing before implementation. Ratschiller and Gerken conclude this in a great line: "Know your enemy - never underestimate him." [Ratschiller]

System development is a wide concept and there are many different views on the system development process. Because of these views, several different methodologies have emerged on how to handle the process. We will start to

introduce one methodology called the Rational Unified Process (RUP) and methodically narrow in on its analysis and design phase where UML often is used.

After presenting UML we will explain WAE that we are focusing on in this thesis.

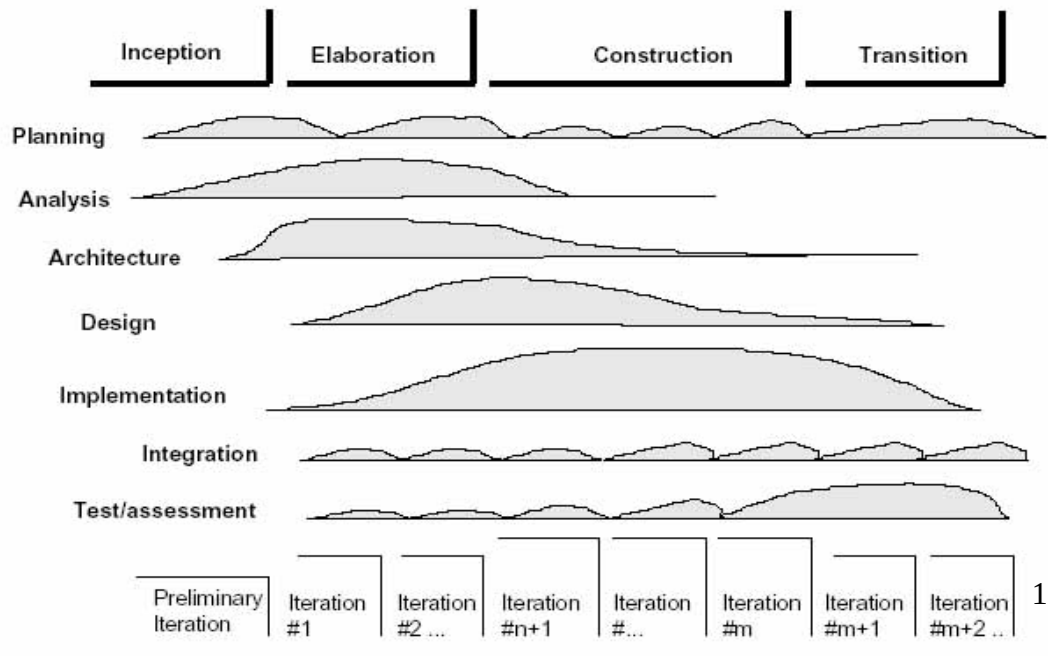
~~5.1~~3.1 Rational Unified Process

The Rational Unified Process (RUP) is a software engineering process developed by Rational Software Company. RUP specifies that it is unrealistic to take a linear approach in a software development process and that work must be done in an iterative manner. The process is strongly use case driven which means that it puts large effort into understanding how the system will be used. [Jacobson]

The Rational Unified Process can be attacked from two perspectives; a management and one technical perspective. The managerial perspective has four phases: interception, elaboration, construction and transition. When you have gone through all the phases you have completed a cycle, which gives a new generation of an executable software, and then the evolution goes on with new cycles.

The technical perspective of RUP is based on iterations and in picture 3.1 below you can see that iterations end simultaneously with phases in the managerial perspective.

To model and visualize the analysis, architecture and design phases RUP users often use UML.



Picture 3.1 – RUP workflow

~~5.1.1~~3.1.1 UML: a model for analysis and design

UML stands for Unified Modeling Language and is a model on how to analyze and design in a system development process. This process can be RUP driven because RUP is especially well suited for the use of UML. [Jacobson] UML uses an Object-Oriented approach and is suited for Object-Oriented programming languages e.g. Java and Delphi.

The history of UML began in 1994 when Grady Booch, the creator behind the Booch model and Jim Rumbaugh, who had created the Object Modeling Technique (OMT), joined their models. Later the three amigos were formed when Booch and Rumbaugh became accompanied by Ivar Jacobson, who earlier had created the Object-Oriented Software Engineering model (OOSE). UML then became adopted as a standard by OMG (Object Management Group) in January 1997.

Object-Oriented Analysis/Design

The use of UML consists of two parts: analysis and design. Analysis is performed to find and describe objects and concepts in the problem domain. The design phase is intended to define objects that will be implemented in an Object-Oriented programming language. [Larman]

A UML process should be:

- "Use case driven"
- "Architecture-centric"
- "Iterative and incremental"

[Jacobson]

This means that use cases are used as a major part of the project e.g. for establishing the systems behavior but also to communicate among the project's stakeholders. The process should use the system's architecture as the primary artifact in the process. An iterative and incremental process releases new and improved executables. The process are also said to be risk-driven which means that the process aims to eliminate the largest risks of the project's success first. [Jacobson]

Goals

The primary goal for modeling with UML is to construct a good application. But to achieve this UML focuses on four points:

Visualize the system

The models will visualize how the system will become. UML specifications include well-defined semantics, which allows one developer to draw a model, and then another can interpret it unambiguously. [Jacobson]

Specifying the system

UML supports "building models that are precise, unambiguous and complete." This means that UML has clear guidelines of how the system's model diagrams will be created and how to interpret them. [Jacobson]

Constructing the system

UML supports both forward and backward engineering. Forward engineering means that you can generate code in an Object-Oriented programming (OOP) language from the diagrams created with UML. Backward engineering is the exact opposite where you can recreate UML diagrams from OOP code. [Jacobson]

Document the system development process

The artifacts that a software development process will produce are requirements, architecture, design, source code, project plans, tests, prototypes and releases. [Jacobson]

Workflow

UML contains nine graphical presentation models:

1. Class diagram: shows a system's static design view with classes, interfaces, collaborations and relationships.

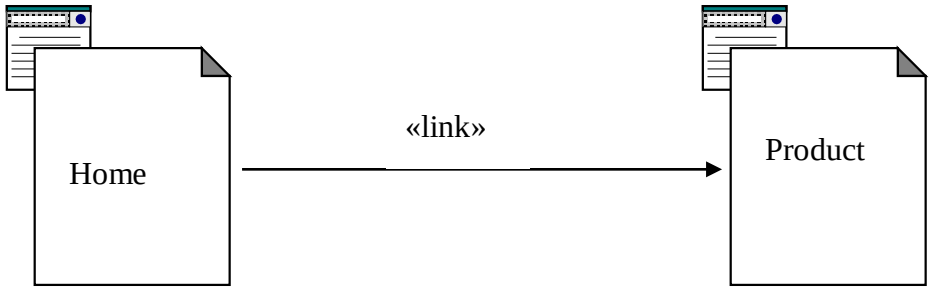
2. Object diagram: shows some objects and their relationships at a particular time.
3. Use Case diagram: shows the actors of the system and their relationships with use cases.
4. Sequence diagram: shows messages between objects.
5. Collaboration diagram: focuses on the structural organization of objects when sending and receiving messages.
6. Statechart diagram: shows the machine state
7. Activity diagram: shows a system's flow between activities in a system.
8. Component diagram: shows organizations and dependencies between components.
9. Deployment diagram: displays the configuration and dependent components of run-time processing nodes.

The aim for using several diagrams is to look at the system from different perspectives. [Jacobson]

~~5.1.2~~3.1.2 Developing Web applications with UML WAE

For the use of UML in a Web application development process Jim Conallen have constructed the WAE. The extension adds stereotypes, constraints and tagged values to be used in the modeling.

A stereotype adds a new value to a model element. Stereotypes are shown with guillements (« ») but can also be in the form of a new icon e.g. the stereotype *Client page* has its own icon but *Link* is shown «link».



Picture 3.2 Shows how a client page (Home) is linked to another client page (Product).

Web application architecture patterns

A pattern describes a core solution to a common problem in the environment. Jim Conallen writes: “An architectural pattern expresses a fundamental structural organization schema for software systems.” [Conallen] There are many patterns available for various situations depending on the technique and architecture, but especially for Web applications Conallen describes the three following architectural patterns. As with all patterns they can all be used on the same architecture, but obviously on different parts and components.

Thin Web client

This pattern is mostly used when there is little knowledge of the client’s environment. That means, just as the patterns name indicate all that the client requires is a browser because all the business logic is executed on the server. This is the most used pattern and is used on Web pages. [Conallen]

Thick Web client

Unlike the thin Web client with no business logic, this pattern is used for clients with a significant amount of business logic executed on its computer. This can be a client-side technique such as DHTML, Java-script, Java-applets, Active X etc. [Conallen]

Web delivery

Situations when the browser acts as a delivery and container device for protocols, instead of only HTTP, such as DCOM, then the Web delivery pattern should be used. That means that it should be used for Web applications that support distributed object systems. [Conallen]

Workflow with UML WAE

This is the workflow of the example *Glossary ASP Application Sample Model*, presented by Jim Conallen in *Building Web Applications With UML*. In this section we will shortly explain in what order the diagrams are used and what the

meaning behind the models are. In Appendix 1 you can find the diagrams from our development to compare with the following diagrams.

Top-Level Use Case View

Use cases are modeled to explain the systems behavior. It focuses on the interaction between the system and its actors. An actor is named a describing name e.g. Online Customer, the use case itself is also named, but here the name will point to the use case's goal. Use cases works with scenarios, there is always one main scenario, but there can be several alternatives. One thing to remember when you work with use cases is that there is no right or wrong, the structure can be everything from the extreme strict to the opposite. [Conallen]

Compare with the use case diagram in the UML chapter (3.1.1).

Analysis model: Main diagram

Use case analysis shows a use case's flow of events and the classes participating. It also shows responsibilities, attributes and associations of the classes. The model has three kinds of stereotyped classes:

- Boundary objects: interface between actor and system
- Entity objects: e.g. Customer, Product
- Control objects: processes, named activities

[Conallen]

Compare with the activity diagram in the UML chapter (3.1.1).

Analysis model: Use Case Sequence Diagram

The sequence diagram is an interaction diagram. It shows the interactions between actor and system like in the use case diagram but now are the timeline emphasized in the showing of messages. One important thing in modeling the sequence diagram is that it is one diagram per use case, no more! [Conallen]

Compare with the sequence diagram in the UML chapter (3.1.1).

Sequence Diagram for the Use Case Realization

This sequence diagram shows the same information in a more object oriented thinking than the first sequence diagram in the workflow. This means that it does not only show methods but also their parameters and return types.

Compare with the sequence diagram in the UML chapter (3.1.1).

Server Components Package - Class Diagram

This diagram shows the methods connecting the application to a database. All the access to the database goes through this package. [Conallen] This is not always a database but contains interfaces to connect to other parts that are needed for the functionality of the application.

Compare with the class and the object diagram in the UML chapter (3.1.1).

Web Pages Package – Class Diagram

This Class diagram consists of two different ways of modeling, the overview and the detailed. The overview displays pages/scripts and the connections between them. The detailed starts from the overview but adds attributes and methods.

Compare with the class and the object diagram in the UML chapter (3.1.1).

Main Component Diagram for Web Pages

This model shows the links between server pages in a navigational structure and displays redirects, links and the like.

Compare with the component diagram in the UML chapter (3.1.1).

64 Evaluating UML WAE

To evaluate UML WAE we have modeled a Web application using UML WAE as modeling language. During the development work we have analyzed the resulting design. The aspects we focused on are how the model solves the maintainability; by this we mean documentation, reusability, and the ability to extend the application.

We have also chosen to use some specific data collection methods collected from the evaluation methodology described in *Utvärderingsboken* by Jan-Axel Kylén. We found these methods to be useful when analyzing our workflow with UML WAE. The specific methods are used to structure observations and interviews. The methods consist of interview guides, reading guides and observation schemes. [Kylén]

6.14.1 Planning

To be able to perform a successful evaluation we had to gain the necessary knowledge about how to model Web applications with UML WAE. This was very important because if we did not do it correct from the start, the end result would not be correct.

Therefore we have read Building Web Applications by Jim Conallen, followed the examples and discussed the various models until we got a good understanding of all the steps and aspects of the extension. Preparing and planning for the evaluation also included making a list of requirements for a fictional Web shop that should function as a subject of examination for UML WAE and serve as a base for the final evaluation.

~~6.1.1~~4.1.1 How to evaluate according to our criteria

Extensibility

To measure the extensibility we have chosen to evaluate coupling and cohesion on the following levels:

- Coupling between the different packages
- Coupling between the different pages inside the packages
- Cohesion inside the different packages

To evaluate coupling we have chosen to do two different evaluations to measure both the "outer" coupling between the different packages and the "inner" coupling between the different pages inside the packages. To measure cohesion we evaluated the cohesion inside the different packages.

We argue that these factors are of significant value to our thesis, because we have modeled the requirements one by one, aiming to create independent packages with high cohesion and low coupling between and inside them.

Coupling

To measure the coupling between the different packages we counted the number of calls made from each package to other packages and then calculated an average number. This average served as a base for evaluating if a package had low coupling. [Stotts]

To measure the coupling between the different pages inside the packages we counted the number of calls made from each page to other pages instead of between packages and then calculated an average number. This average served as a base for evaluating if a page had low coupling. [Stotts]

Cohesion

We assessed the cohesion inside the different packages by giving the current package a score on a scale from 1 to 9 where 1 is the lowest, indicating low cohesion. If an examined package handled many unrelated tasks it was awarded

with the score 1, but a package that handled a single and unified task received a 9, which indicated high cohesion. All packages are assessed and given a number on the scale mentioned above and an average was calculated. [Stotts]

Generalization/specialization hierarchy

To measure this principle we investigated if there were any generalization/ specialization hierarchies and assessed if these attempts supported extensibility by providing an easily extendable hierarchy. To assess the attempts we used the following rule:

Is-a Rule: This rule can be used as an informal test by forming the statement "Subtype is a Supertype." [Larman] E.g. MySQL_DBConnection **is a** DBConnection, where DBConnection is a supertype and MySQL_DBConnection is a subtype of DBConnection. The result is presented in a simple Yes/No manner.

Reusability

Software reusability can be measured in the term of Lines of Codes (LOC). Poulin uses the following formula to calculate reuse in percent:

$$\frac{\text{Reused Software (LOC)}}{\text{Total Software (LOC)}} * 100$$

[Poulin]

As we have not implemented the design we could not calculate a value of how much code that could be reused. Instead we measured reusability in two slightly different ways. First what packages that could be reused and second what pages that could be reused, by either this or another application.

Furthermore, the measurement was divided into black-box and white-box reusability also according to Poulin. With black-box reuse we do not tolerate any changes to the examined object. When measuring white-box reuse, we took into calculation an estimate of the amount of modifications needed, to fit the examined object to a new environment. Below follows the description of the formulas that were used to produce a result in percent:

Black-box

$$\frac{\text{Reused Packages}}{\text{Total No. of Packages}} * 100$$

$$\frac{\text{Reused Pages}}{\text{Total No. of Pages}} * 100$$

White-box

$$\frac{\text{Reused Packages}}{\text{Total No. of Packages}} * 100$$

$$\frac{\text{Reused Pages}}{\text{Total No. of Pages}} * 100$$

Documentation

Diagrams

The diagrams should not be complex to read, the following criteria marks out good documentation:

- Small detailed diagrams
- Large non-detailed diagrams
- Not written down semantics of each method on a class-by-class basis.
- Good traceability

[Booch]

Traceability

Traceability is defined as the ability to describe and follow a requirement through the design. We have tested our documentation by the following criteria:

- Trace forward from requirements
- Trace backward to requirements [Jarke]

All the requirements were traced from the initial top-level use case view diagram and all the way through the various diagrams. Then we traced the requirements backwards through the diagrams. We observed if the name and meaning of the requirement remained unchanged, if the context was intact and if the transitions between the chains of diagrams were understandable and easy to follow.

We have evaluated all the documentation criteria and assessed them by the following grades:

F: Failed to support
G: Good support
VG: Very good support
*: Could not be measured

~~6.1.2~~4.1.2 Participants profile

Because we had problems to even find companies that used models to design their Web applications, we did not have any expectations to find any companies to conduct our evaluation at. This made us to decide that we would carry out a development process, using UML WAE, ourselves and evaluate it. The decision was based on the estimation of our own knowledge and ability to perform a development process in a correct way.

With this approach come certain risks, such as the difficulty to criticize the own work and separate between the development and evaluation process. We are aware of these risks and as an attempt to avoid them we have divided the work into two roles, developer and evaluator. The evaluator's role is mainly to supervise the developers' work and control that the development process is conducted in a professional manner. The developers' roles are to design the application without consideration that the result is going to be examined afterwards.

The development team consisted of two persons. Both of the developers have studied software engineering, with computer science as their major, for three years or more. One of them has worked as a student assistant in Object-Oriented programming and UML, the other is self-employed and manages a Web based golf portal.

The evaluator has also studied software engineering, with computer science as a major, for three years and has worked as a student assistant in Object-Oriented programming. However, he has no prior experience with evaluating so the evaluation knowledge is restricted to what we could find in the literature.

When proceeding to the analysis of the result we left our previous roles as developers and evaluator. All the participants now took the role as evaluator and worked together to analyze the resulting design material.

All of the participants have prior knowledge of UML for non-Web application development from courses taken at Blekinge Institute of Technology. We have also experience of Web application development such as Web shops.

~~6.1.3~~4.1.3 Data collecting

Diagrams

The diagrams that the modeling with UML WAE produced were our main source of information. The diagrams were produced during the design sessions. Then we discussed and revised them before we translated them into digital form. All the conclusions in the result originate from these diagrams. See Appendix 1.

Observations and interviews

The data collecting of information through observations and interviews was mainly the evaluator's responsibility. He observed the developers during the design sessions and conducted frequent interviews with the developers. These frequent "controls" were made to inspect that the design process was conducted properly. To his help he had two different tools: observation schemes and interviews.

Observation schemes

The developers filled in diaries after every design session and handed them in to the evaluator at the end of every week. The diaries were designed in a form that made them useful in the analysis of UML WAE. The evaluator used the diaries to raise questions that he could use in the interviews. Furthermore, the evaluator participated on every design session and observed the work and the developers. An example of a diary used can be found in the Appendix 2 (in Swedish).

Interviews

The interviews were conducted in two forms: interviews performed during the design work and reviews performed after the design work was completed. The interviews performed during the work occurred one time every week. The evaluator used the issues brought up in the diaries to construct interview guides that he could follow during the interview. After the design work was complete a review was held with the entire group to discuss the process and result.

The structure of the diaries and main structure of the interviews was constructed by us together but the evaluator determined their contents.

~~6.1.4~~4.1.4 Case study

The Web shop we conducted the design with UML WAE on is a basic online shop where customers can browse and buy products. There are also functions included to support administration and order handling. The case study starts from five basic requirements and finishes before the implementation phase.

List of requirements

We have categorized and specified the Web shops functions in the five following groups:

Show products

Information: Product database

- Ability to list the products by their category
  - Listing with name and price
  - Listing with name, full description, price and image
- Ability to perform searches based on your own search criteria

Shopping Cart

Information: Product database

- Ability to add a product to the cart
- Ability to remove a product from the cart
- Ability to change the number of products in the cart

Check out point

- Ability to register an order
- Ability to supply the address details
- Ability to choose payment method
- Ability to choose fright/toll method

Administration

- Ability to add products
- Ability to change/edit products
- Ability to remove products
- Ability to logon
- Ability to logout

Order handling

- Ability to show orders
- Ability to change/edit orders
- Ability to logon
- Ability to logout

~~6.2~~4.2 Conducting analysis and design

When the requirements for the Web shop and the UML WAE preparations were made we conducted the modeling phase, which then would be the foundation of our evaluation and result.

~~6.2.1~~4.2.1 Design sessions

The design sessions were carried out in small group rooms equipped with one table, six chairs and a whiteboard. The sessions stretched for eight hours, per session, with one-hour lunch breaks. Both developers and the evaluator participated in all the design sessions.

The developers took turns to sketch the diagrams on the whiteboard and to write them down on paper. One of the developers then had the responsibility to transmit the diagrams into digital form. While the developers worked on the design the evaluator observed and took notes.

~~6.2.2~~4.2.2 Workflow

The developers decided to perform the design on one requirement at the time, and follow that requirement through the entire development cycle before starting with the next. The diagrams that visualize overviews over more than one requirement were modeled last because we needed all the requirements to model the overview. This workflow is similar to the iteration cycles that RUP uses.

Another decision that the developers took was to design according to the *thin client* pattern mentioned in chapter 3.1.2 Developing Web applications with UML WAE. This decision was made because the developers did not have any knowledge of what platforms the clients will run the application on.

For every step in the process with the different diagrams the developers discussed the situation and argued for different solutions. This led to small disputes but we believe that it helped the process in a positive manner. If no questions had been raised then the work would not have reached the same quality. We believe that questioning and discussing the diagrams led to a better result than if we had agreed the whole time.

When all the diagrams were designed and completed we started to examine them with our predetermined criteria. We started to examine the extensibility followed by the reusability and finished with the documentation criteria. Applying our metrics on the diagrams produced the result and we used all the diagrams in the review process, to get the most accurate result possible.

7 Result on evaluation of UML WAE

The presentation of the result is divided into the different criteria. Each criteria is presented with data, analyze of data and a sub conclusion. The sub conclusions are then analyzed and the final conclusion is derived.

7.1 Extensibility

Coupling between the different packages

Package:	**"Outer" couplings:**
Login/out	2
Add products	2
Search	3
Edit	3
Database	4
Login/out	1
Order	3
Database	2
Homepage	5
Show categories	2
View products	5
Cart	5
Search	5
Checkout	2
Database	4

Total outer couplings: 14 + 6 +28 = 48
Packages: 15
Outer couplings per package: 48/15 = **3,2 couplings/package**

Coupling between the different pages inside the packages

The inner couplings column specifies how many couplings a page has to other pages. E.g. there are nine pages in our example that are only connected to one other page.

Pages:	**"Inner" couplings:**
9	1
16	2
7	3
6	4
6	5

Total inner couplings: 1*9+2*16+3*7+4*6+5*6=116
Pages: 9 + 16 + 7 + 6 + 6 = 44
Inner couplings per page: 116 / 44 = **2,6 couplings/package**

Cohesion inside the different packages

Package:	**Result:**	**Short motivation:**
Login/out	5	Some other couplings
Add products	7	Handles its assignment
Search	8	Performs its duty and pass on the result
Edit	6	Uses an outside package
Login/out	7	Good but slightly small
Order	7	Handles its assignment
Homepage	3	Controller pattern
Show categories	6	Handles its assignment
View products	5	Depends on other packages
Cart	7	Large, but self-supporting
Search	4	Depending on view products
Checkout	7	Self-supporting

Overall package cohesion

5+7+8+6+7+7+3+6+5+7+4+7 = 72 total result
12 packages
72/12 = **6 average cohesion result**

Generalization/specialization

Is-a rule

AdmProductCatalog **is a** DBHandler
Staff **is a** DBHandler
OrderCatalog **is a** DBHandler
ProductCatalog **is a** DBHandler

Although the names could have been more thought through the **Is-a** rule apply; all the subclasses are DBHandlers.

~~7.1.15~~5.1.1 Analyzing Extensibility

Low coupling

Our design resulted in low coupling with an average of 3,2 outer couplings/package and 2,6 inner couplings/page. We argue this because with a maximum of five couplings between the measured items and averages around three we definitely consider the result to be low coupling both between packages and the pages.

High cohesion

We assessed the cohesion in our diagrams to have the average 6 on the scale from 1 to 9. This is considered to be over the average and therefore high cohesion. The packages we created were small and specialized on their part of the system and were developed only to handle this.

Generalization/specialization

Only the general database connection class with its methods was used as a superclass with subclasses handling specialized tasks specific for its package. UML WAE supported modeling with hierarchies so the result is simply: Yes.

In what way does UML WAE support Extensibility?

UML WAE supports Extensibility by the possibility to design for low coupling, high cohesion and the creation of generalization/specialization hierarchies.

~~7.2~~5.2 Reusability

Total numbers of pages: 42
Total numbers of server pages: 15
Total numbers of packages: 15

Black-box reuse

Packages $\frac{0}{15} * 100 = 0\ \%$

Pages $\frac{0}{42} * 100 = 0\ \%$

White-box

Packages $\frac{5}{15} * 100 = 33\ \%$

Pages $\frac{15}{42} * 100 = 36\ \%$

~~7.2.1~~5.2.1 Analyzing Reusability

Black-box reuse

The strict rules for black-box ended in 0% reuse both for the packages and the pages. However, this was almost expected because of the large effort needed to achieve black-box reuse.

White-box reuse

The white-box reuse seen over all the 15 packages were estimated to be 5 of these and were calculated to 33%. In the 5 packages that we estimated could be reused only small modifications have to be made, such as modifying or changing pages that contain domain specific functionality. The other 10 demanded too much alteration to be considered for reuse.

The white-box reuse of the pages was calculated to 36%. From all the pages (42 pages) we calculated that all the server-pages (15 pages) could be reused on a similar application with a small amount of modifications. The changes that have to be done to the server-pages are of smaller nature like changing variable names and links to Web and server pages. The server pages have standardized functions and are easily transformed to a different Web shop, if the programmer who programmed them from the beginning did a good job.

The percentage 33% respectively 36%, is mainly a result of the application's none reusable client-pages. If the number of none reusable client-pages had been smaller the percentage had been higher.

In what way does UML WAE support Reusability?

UML WAE supports Reusability due to the ability to white-box reuse. This thesis was unable to clarify if UML WAE supports reusability regarding black-box reuse.

~~7.3~~5.3 Documentation

Diagram	Small detailed diagrams	Large non-detailed diagrams	Not written down semantics
Top-Level Use Case View	G	*	G
Use Case Analysis	G	*	VG
Use Case Sequence Diagram	G	*	G
Sequence Diagram	VG	*	G
Overview Class Diagram	*	VG	G
Detailed Class Diagram	*	F	G
Main Component Diagram for Web Pages	*	VG	G

Traceability

Grade: VG

~~7.3.1~~5.3.1 Analyzing Documentation

UML WAE's different diagrams generally got good grades based on our criteria when evaluating the diagrams. They followed our guidelines with small detailed diagrams, large non-detailed diagrams and not written down semantics with a few divergences, but overall UML WAE obtained a grade slightly above G.

The models created with UML WAE are based on its function and grouped together in a naturally form starting with simple use case diagrams and ending with detailed class diagrams.

The model includes a large non-detailed diagram. This overview diagram shows an overview over the whole system and serves as a map for the programmer. It was easy to read and if you have studied the previous diagrams you can easily find where the functions are.

The detailed class diagram got a fail from us because it is large and detailed. But that mark was based on our criteria and can be discussed. Before looking at the detailed diagram you must have gone through the overview diagram closely before so you can see the whole picture.

The UML WAE describes the application with non-written semantics, which means that the pages/functions and their relationships are expressed in visual diagrams, rather than in plain text.

This approach lowers the amount of unused documentation because written documentation of semantics often remains unread. It is easier to get an overview and understanding when looking at a whole diagram with the coupling, relationship etc. rather than in writing.

Traceability

The given grade VG was our overall opinion of the diagrams' traceability. The UML WAE supplied good traceability in the documentation because the method allowed the participants to get a grip on the functions of the coming software. The method started with broad and easy models such as use cases that then naturally and in a gradually manner transformed into more complex models. This made a good transition from requirements to the final class diagram that was easy to follow and develop from.

In what way does UML WAE support Documentation?

UML WAE supports Documentation by producing understandable, easy to read and highly traceable diagrams.

~~7.4~~5.4 Conclusion

The use of UML for Web application development will result in good design, regarding to maintainability.

Extensibility was supported through low coupling, high cohesion and the possibility to create generalization/specialization hierarchies.

Reuseability was supported by the ability to apply white-box reuse. This thesis was unable to resolve whether UML WAE supports black-box reuse.

The produced documentation was understandable, easy to read and had highly traceable diagrams.

With our three criteria extensibility, reusability and documentation examined and found to be in support, the conclusion is that the use of UML for Web application development resulted in good design regarding to maintainability.

~~8~~6 Discussion

While writing this thesis several related issues, which did not quite fit in the report, came up to discussion. Below these issues are discussed, but first our questions and thoughts concerning our method and the validity and reliability of the result are presented.

~~8.1~~6.1 Validity and reliability of the investigation

Due to the lack of an appropriate investigation environment to conduct our thesis in, we chose to perform a small design project of our own. The decision was taken based on believes that we had the experience and knowledge to conduct a design project in a professional matter on our own.

We chose to design a simple Web shop to test our selected criteria on. The design of the Web shop took valuable research time and we had to limit the extent of the example. This may have influenced the final result because of the limited test material.

A better solution to this would be to perform our investigation on a real ongoing design project where the developers use UML WAE. Then our focus could have been on collecting the research data and evaluating it, and not spending valuable time on designing.

Furthermore, our requirements are not formed exactly according to existing methods. This is also a decision we took to save time. However, the requirements are well thought through and serve their purpose and we do not think they affected the result in a negative manner.

Concerning the validity of the result from the case study, we believe that the maintainability, which was our goal to measure, was measured in a correct way. This is stated on our criteria, and the metrics we used to measure them. Most of the metrics were collected from different sources that handled object-oriented design and most of them could be used without alterations in our case study. However, some of them had to be slightly altered to function in our case study but we believe that these alterations did not affect the result.

~~8.2~~6.2 Model for Client/Server but not for the Web

There is a major difference between a Web application and a Web site, but not between a Web application and a Client/Server application. The paradigm of Web applications is changing and the boundaries between Web applications and Client/Server application are becoming less distinct. When examining the underlying techniques and architectures you will see that many of the same components exist and the ideas are the same. That is why it is strange that modeling Client/Server applications is standard but not when developing Web applications.

This following answer we got in the pre-study from one Web developer sums up this old attitude towards modeling Web applications: *It is only to throw in the necessary components and make it work*. So even now when Web sites are much more like applications this mentality still seems to exist.

~~8.3~~6.3 Education

As pointed out above, we believe the distinction between a Web application and a Client/Server application is vague. That is why we question why modeling when developing applications for the Webb is so unheard of, but when developing Client/Server applications modeling is obvious.

One answer to that question could be that universities only teach their students to model for Client/Server applications, not for Web applications. For instance our own school BTH is a good example of this; UML and various other modeling techniques such as ER-modeling have always been taught, used and carefully examined in previous courses. But when we implemented a Web shop in HTML and PHP with a MySQL database in the course “Web technologies” modeling was barely mentioned.

Internet and its use with new Web applications of various forms, is still young and are evolving every day. There are new techniques and standards for the Web emerging constantly, so perhaps the computer science faculties at the universities have not kept up with the pace?

To consider the development of Web applications equally complex and important as the development of Client/Server applications seems more up to date.

~~8.4~~6.4 Is UML WAE the best choice?

In many cases we felt that the UML WAE itself neither supported nor overturned some aspects.

For an example UML WAE will not provide its user with a reusable result; it is more up to the developers to create reusable components. Conallen's book does not even mention reusability so the UML WAE workflow will not automatically provide for it, but at the same time it will not prevent it either. If you as a developer have knowledge about reusability you have the ability to create reusable code with the UML WAE notation.

The same goes for low coupling and high cohesion. In his book Conallen advocate both low coupling and high cohesion as good design patterns and he claims usage of his model can contribute to achieve this. It is however up to the developers designing the application to apply these patterns; the model itself can't provide it but only support and allow it, which UML WAE does.

Although UML WAE did not have all the features we might have wanted, it did not either limit us so our overall judgment is positive. To say that UML WAE is the best choice is impossible for us to say since we have not compared it with other similar modeling languages, but it definitely is a good tool when modeling for the Web.

~~8.5~~6.5 Future studies

Interesting ideas for future studies in this area might be:

- To follow a "real" case study conducted by Web developers using UML WAE
- To compare UML WAE with other similar models

9 7 Literature

Booch, Grady. *Object-oriented analysis and design with applications*. Addison Wesley, 1994.
ISBN: 0-8053-5340-2

Conallen, Jim. *Building web applications with UML*. Addison Wesley, 2000
ISBN: 0-201-61577-0

Eklund, Sven, and Hans Fernlund. *Programkonstruktion med kvalitet – projekthantering och ISO 9000*. Lund: Studentlitteratur, 1998.
ISBN: 91-44-00626-8

Fournier, Roger. *A Methodology for Client/Server and Web Application Development*. Prentice Hall, 1999.
ISBN: 0-13-598426-2

Gillies, Alan C. *Software Quality*. Chapman & Hall, 1992.
ISBN: 0-412-45130-1

Hernbäck, Jan, et al. *Systempraktikan – en handbok i systemanalys*. Malmö: Liber, 1990.
ISBN: 91-40-30927-4

Jacobson, Ivar, Booch, Grady, and Rumbaugh, James. *The Unified Modeling Language user guide*. Addison Wesley, 1999.
ISBN: 0201571684

Kylén, Jan-Axel. *Utvärderingsboken*. Stockholm : Kylén, 1992.
ISBN: 9185652504

Larman, Craig. *Applying UML and patterns*. Prentice-Hall, 1998
ISBN: 0-13-748880-7

Poulin, S., Jeffrey. *Measuring Software Reuse*. Addison Wesley, 1997.
ISBN: 0-201-63413-9

Powell, Thomas A. *Web design: the complete reference*. Osborne/McGraw-Hill, 2000.
ISBN: 0-07-212297-8

Ratschiller, Tobias and Gerken, Till. *Web application development with PHP 4.0*. Prentice Hall, 2000.
ISBN: 0-7357-0997-1

~~9.1~~7.1 Web sources

Gotel, Orlena Ph.D. Contribution Structures for Requirements Traceability. London, England: Imperial College, Department of Computing, 1995. Available online 2002-05-12 at: http://src.doc.ic.ac.uk/public/ic.doc/dse/viewpoints/olly_phd_thesis.ps.gz

Greene, Patrick. Available online 2002-03-06 at http://www.xgenapplications.com/pro-con.htm published in January 2001

Jarke, Matthias Ph.D. Available online 2002-05-12 at http://portal.acm.org/citation.cfm?doid=290133.290145 Published December 1998 in Communications of the ACM (Volume 41, Issue 12)

Kafura, Dennis Ph.D. Available online 2002-04-26 at http://people.cs.vt.edu/~kafura/cs2704/oop.swe.html Published in June 1996

Royce, Walker. Available online 2002-04-15 at http://www.therationaledge.com/content/feb_02/f_conventionalToModern_wr.html Published February 2002 in *The Rational Edge*

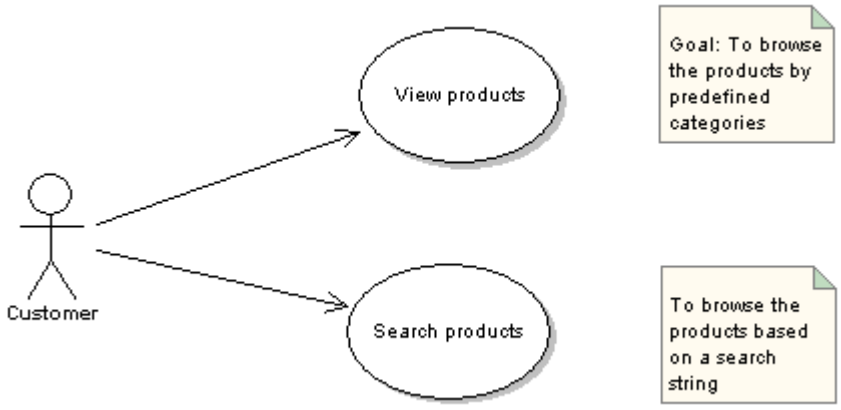
Stotts, David. Associative Professor. University of North Carolina. Available online 2002-05-13 at: http://www.cs.unc.edu/~stotts/145/homes/audio/techman/desanal/design_analysis.html

Sullivan, John Available online 2002-03-06 at http://searchnetworking.techtarget.com/sDefinition/0,,sid7_gci211796,00.html published Jul 29, 2001

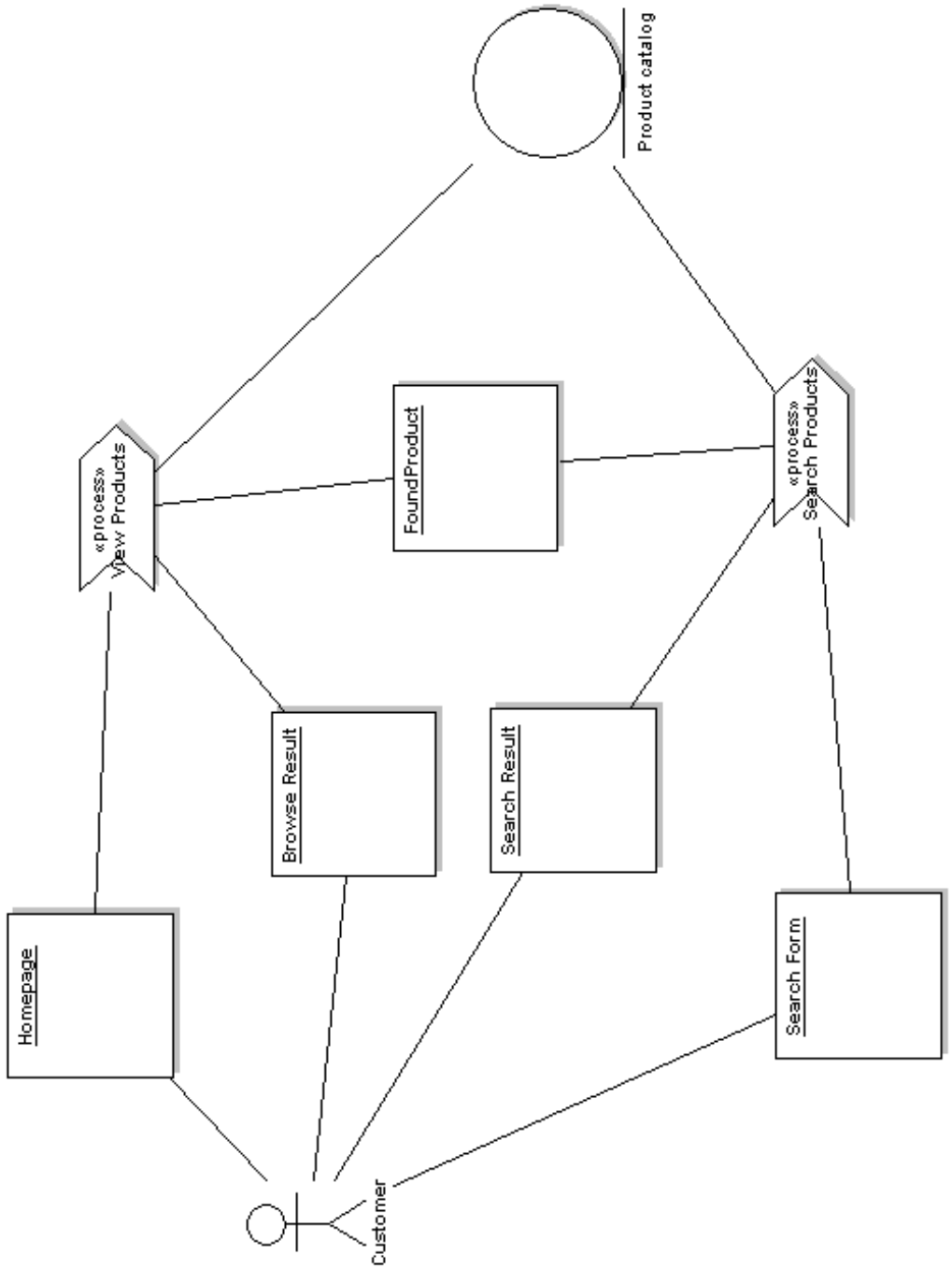
Appendix 1

The following is some design material derived from the work with UML WAE.

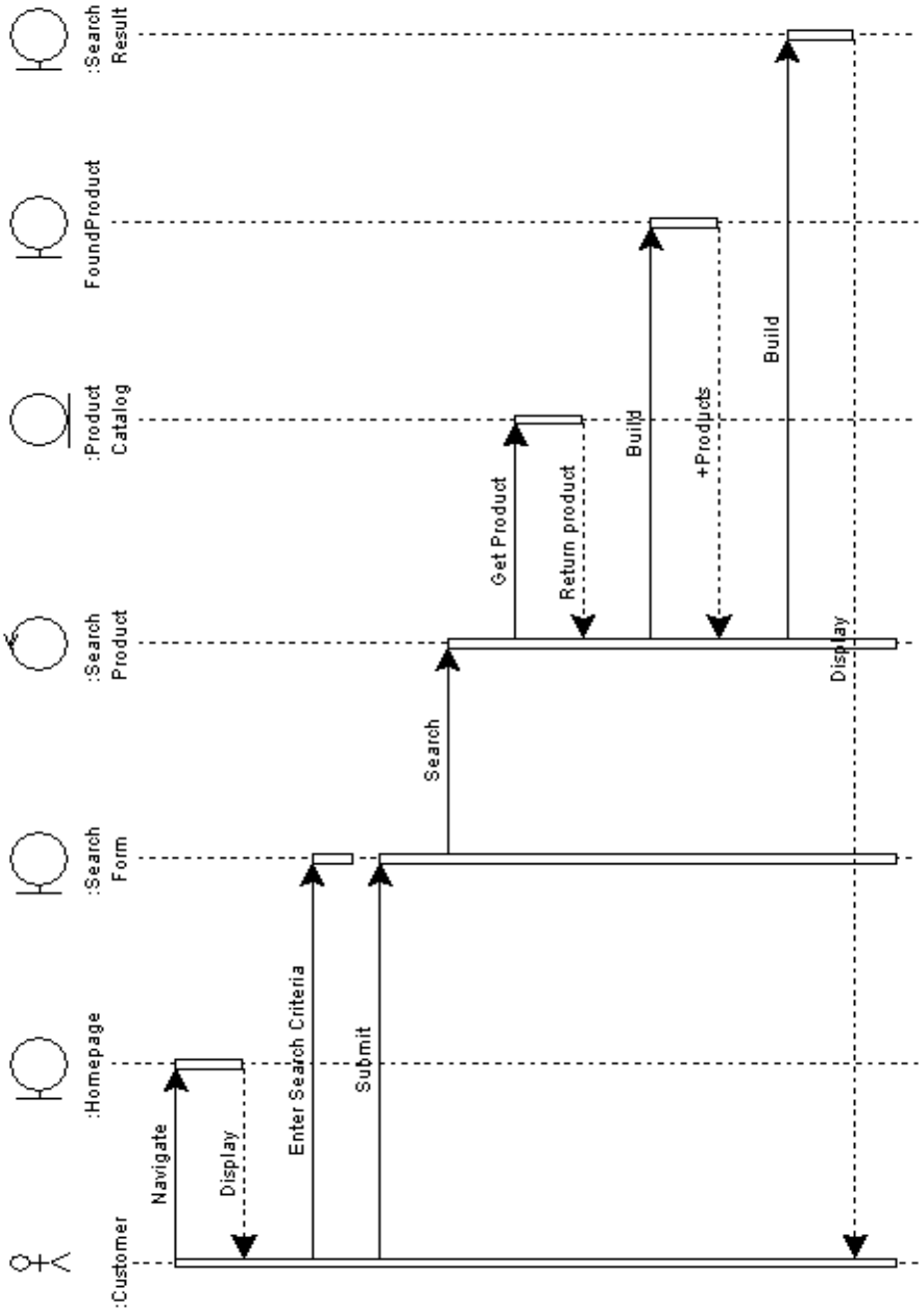
Top-Level Use Case View



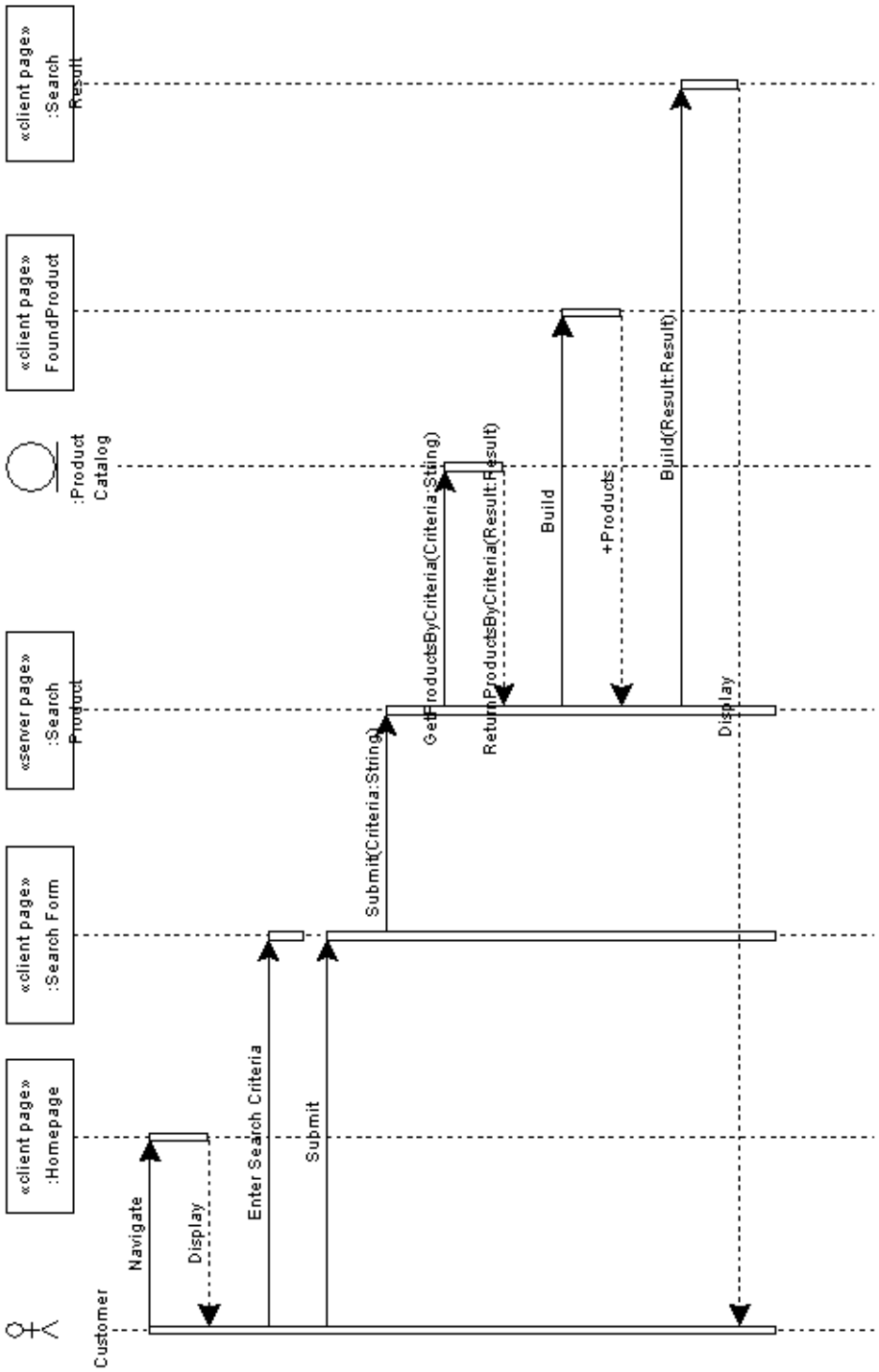
Analysis model: Main diagram



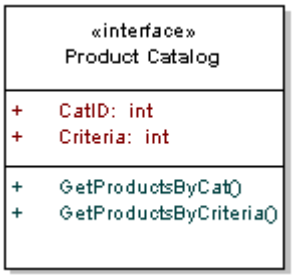
Analysis model: Use Case Sequence Diagram



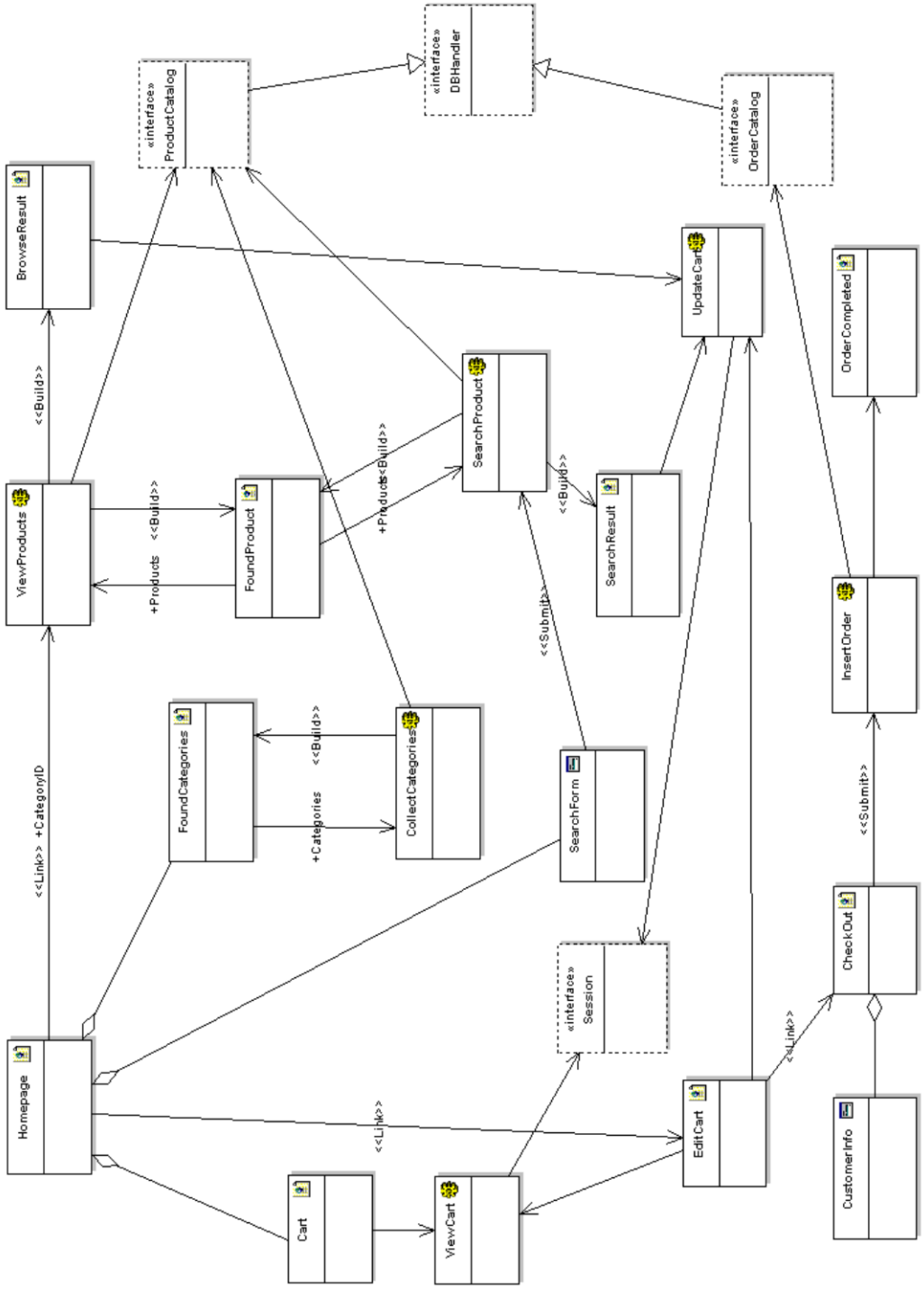
Sequence Diagram for the Use Case Realization



Server Components Package - Class Diagram



Web Pages Package – Class Diagram



Appendix 2

Dagbok utvecklingsarbete

Datum: Vecka:
Namn:

1. Arbetsuppgift/uppgifter denna dag (Vilket/vilka diagram):

2 Vad har gått bra med tanke på modellens funktion och dina arbetsuppgifter?

2.1 Vad tror ni det beror på?

3 Vad har gått dåligt med tanke på modellens funktion och dina arbetsuppgifter?

3.1 Vad tror ni det beror på?

3.2 Hur skall det kunna undvikas/minskas?

4. Övriga observationer/anmärkningar?

Att tänka på maintainability, extensibility, reusability och documentation och begränsningar i modellen.

OBS. Dagböckerna skall vara inlämnade till Tobias senast Fredag varje vecka antingen skriftligt eller via mail! Dokumentet finns på ftp-servern.