Final Thesis

A Usability Engineering approach to developing an e-commerce web application

by

Markus Kvist

LITH-IDA-EX--06/070--SE

2006-10-23

Linköpings universitet
Department of Computer and Information Science

Final Thesis

A Usability Engineering approach to developing an e-commerce web application

by

Markus Kvist

LITH-IDA-EX--06/070--SE

2006-10-23

Supervisor: Peter Boqvist
StjärnDistribution AB

Examiner: Lena Strömbäck
Institutionen för Datavetenskap
Linköpings universitet

Abstract

Usability is an important aspect of product development that deals with the quality of interaction between the user and the product. Capturing requirements is aimed at understanding what the users expect the system to do. This report attempts to apply principles from both the Usability Engineering and the Requirements Engineering methodologies in the development of a subsystem for managing an e-commerce solution. The main focus is on the user interface design, which is developed by iterating the three phases of design, usability testing and evaluation, until the desired level of usability is achieved. Prototyping was central to the rapid development, but the use of software prototyping tools instead of paper-mockups would reasonably have improved the usefulness of the evaluations. The desired usability level was set to reach a specified fix point, instead of attempting to achieve some absolute usability metrics as is typically practiced. The former approach was apparently simpler to use without the broader experience otherwise needed to write reasonable requirements. The transition from the throw-away prototype to the implemented evolutionary prototype in ASP.NET posed some interesting problems that are further discussed.

Table of contents

LIST OF FIGURES ....

LIST OF TABLES ....

1 INTRODUCTION 1

1.1 BACKGROUND 1

1.2 PURPOSE 1

1.3 PROBLEM DESCRIPTION 1

1.4 SOLUTION METHOD 1

1.5 SCOPE 1

1.6 SOURCES 2

1.7 STRUCTURE 2

1.7.1 Outline 2

1.7.2 Abbreviations 2

1.7.3 Glossary 2

2 PROBLEM DESCRIPTION 4

2.1 OVERVIEW 4

2.2 EXISTING SYSTEM 4

2.2.1 The web shop's front-end 4

2.2.2 The web shop's back-end 5

2.3 SUGGESTED SOLUTION 6

3 METHODOLOGY PREREQUISITES 8

3.1 SOFTWARE ENGINEERING 8

3.1.1 Requirements Engineering 8

3.1.2 Design 10

3.1.3 Testing 12

3.2 USABILITY ENGINEERING 13

3.2.1 Introduction 13

3.2.2 User profiling 15

3.2.3 Task analysis 15

3.2.4 Usability metrics 16

3.2.5 Evaluating Interface Designs 16

4 TECHNICAL PREREQUISITES 19

4.1 MICROSOFT .NET 19

4.1.1 .NET Framework 19

4.1.2 ASP.NET 21

4.2 DATABASE DESIGN 23

4.2.1 Introduction 23

4.2.2 Data modelling 24

4.2.3 Relations *25*
4.2.4 Types of keys *25*
4.2.5 Normalization *26*
4.2.6 Synthesis of relations *26*
4.2.7 Database design *27*
5 CAPTURING REQUIREMENTS **29**
5.1 REQUIREMENTS GATHERING 29
5.1.1 Purpose *29*
5.1.2 Method *29*
5.1.3 User Questionnaire *29*
5.2 REQUIREMENTS SPECIFICATION 32
5.2.1 Sources of requirements *33*
5.2.2 Summary *33*
6 INTERFACE DESIGN **36**
6.1 METHOD 36
6.2 OVERVIEW 36
6.3 ITERATION 1 37
6.3.1 Design *37*
6.3.2 Usability testing *41*
6.3.3 Evaluation *43*
6.4 ITERATION 2 44
6.4.1 Design *44*
6.4.2 Usability testing *45*
6.4.3 Evaluation *46*
6.5 ITERATION 3 48
6.5.1 Design *48*
6.5.2 Usability testing *49*
6.5.3 Evaluation *52*
6.6 ITERATION 4 53
6.6.1 Design *53*
6.6.2 Usability testing *54*
6.6.3 Evaluation *55*
7 FINAL DESIGN & IMPLEMENTATION **57**
7.1 METHOD 57
7.2 MAIN ARCHITECTURE 57
7.3 CLASS OVERVIEW 58
7.3.1 Web forms *58*
7.3.2 Utility library *59*
7.4 FINAL INTERFACE DESIGN 59

7.4.1 CheckList *60*
7.4.2 Categories *61*
7.4.3 SDCategories *62*
7.4.4 SDSearch *63*
7.4.5 SDProducts *64*
7.4.6 Associations *66*
7.4.7 Products *67*
7.4.8 ProductDetails *68*
7.4.9 Upload *69*
7.4.10 Pages *70*
7.4.11 Page *71*
7.4.12 Campaign *72*
7.5 STATIC DESCRIPTION 73
7.6 DYNAMIC DESCRIPTION 75
7.7 DATABASE STRUCTURE 76
7.7.1 Database design example: Products *77*
8 DISCUSSION 79
8.1 SYSTEM 79
8.2 METHOD 79
8.2.1 Usability Engineering *79*
8.2.2 Software engineering *81*
8.3 IMPLEMENTATION ISSUES 82
8.3.1 Static variable scope *82*
8.3.2 Grouping with GridView *82*
8.3.3 Grouping with Repeater *83*
8.3.4 Hiding columns in a GridView *83*
8.3.5 Verifying input and deletion in GridView *83*
8.3.6 Container name length *83*
8.3.7 Multiple paths with referential integrity actions *84*
8.3.8 Security considerations *84*
9 CONCLUSION 86
REFERENCES 87
WEB REFERENCES 87
APPENDIX A – QUESTIONNAIRES 88
A.1 USER QUESTIONNAIRE 88
A.2 USER SATISFACTION QUESTIONNAIRE 92
APPENDIX B – EIGHT GOLDEN RULES OF INTERFACE DESIGN 96
APPENDIX C – REQUIREMENTS SPECIFICATION 97
C.1 DOCUMENT STRUCTURE AND DEFINITIONS 97

Document structure *97*
Requirement structure *97*
Priority levels *97*
C.2 USERS OF THE SYSTEM 97
Client *97*
Super Administrator *97*
C.3 FUNCTIONAL REQUIREMENTS 98
Product related requirements *98*
Customer related requirements *102*
Price related requirements *103*
Administration requirements *104*
Statistics related requirements *105*
General requirements *107*
C.4 NON-FUNCTIONAL REQUIREMENTS 107
Usability *107*
Performance *108*
Compatibility with existing environment *109*

List of figures

FIGURE 2.1: OVERVIEW – EXISTING SYSTEM 4
FIGURE 2.2: SAMPLE WEB SHOP OF EXISTING SYSTEM 5
FIGURE 2.3: BACK-END OF EXISTING SYSTEM 6
FIGURE 2.4: OVERVIEW – SUGGESTED SYSTEM 6
FIGURE 3.1: USABILITY ENGINEERING ACTIVITIES 15
FIGURE 4.1: MICROSOFT .NET FRAMEWORK 20
FIGURE 4.2: WEB FORMS AND CODE BEHIND FILES 21
FIGURE 4.3: COMPONENTS OF A DATABASE SYSTEM 24
FIGURE 5.1: AGE AND SEX DISTRIBUTION 31
FIGURE 5.2: COMPUTER EXPERIENCE 32
FIGURE 6.1: DRAFT – SDPRODUCTS (VIEWING SEARCH RESULTS) 38
FIGURE 6.2: DRAFT – PRODUCTS 39
FIGURE 6.3: DRAFT – SDCATEGORIES 40
FIGURE 6.4: ANSWER DISTRIBUTION – USER SATISFACTION QUESTIONNAIRE 1 43
FIGURE 6.5: ANSWER DISTRIBUTION – USER SATISFACTION QUESTIONNAIRE 2 47
FIGURE 6.6: ANSWER DISTRIBUTION – USER SATISFACTION QUESTIONNAIRE 3 52
FIGURE 7.1: MI2 ARCHITECTURE 57
FIGURE 7.2: FINAL DESIGN – CHECKLIST 60
FIGURE 7.3: FINAL DESIGN – CATEGORIES 61
FIGURE 7.4: FINAL DESIGN – SDCATEGORIES 62
FIGURE 7.5: FINAL DESIGN – SDSEARCH 63
FIGURE 7.6: FINAL DESIGN – SDPRODUCTS 65
FIGURE 7.7: FINAL DESIGN – ASSOCIATION 66
FIGURE 7.8: FINAL DESIGN – PRODUCTS 67
FIGURE 7.9: FINAL DESIGN – PRODUCTDETAILS 68
FIGURE 7.10: FINAL DESIGN – UPLOAD 69
FIGURE 7.11: FINAL DESIGN – PAGES 70
FIGURE 7.12: FINAL DESIGN – PAGE 71
FIGURE 7.13: FINAL DESIGN – CAMPAIGN (ALTERNATIVES) 72
FIGURE 7.14: FINAL DESIGN – CAMPAIGN (COMPLETION) 73
FIGURE 7.15: CLASS DIAGRAM – SDPRODUCTS 74
FIGURE 7.16: SEQUENCE DIAGRAM – SDSEARCH AND SDPRODUCTS 76
FIGURE 7.17: RELATIONSSHIPS – PRODUCTS 78
FIGURE 8.1: RESULT COMPARISON – USER SATISFACTION QUESTIONNAIRE 80

List of tables

TABLE 6.1: OVERVIEW – PAGES 36
TABLE 7.1: OVERVIEW – WEB FORMS 58
TABLE 7.2: OVERVIEW – UTILITY LIBRARY 59
TABLE 7.3: CLASS MEMBERS – SDPRODUCTS 74
TABLE 7.4: CLASS METHODS – SDPRODUCTS 75

TABLE 7.5: DATABASE TABLES – PRODUCTS 78

1 Introduction

This chapter briefly introduces the problem and solution method. Chapter 2 explores the problem further.

1.1 Background

The company Star Distribution (SD) is located in the northern part of Sweden. It started in the year of 1994 and currently has about ten employees. The company distributes books and music from different publishers, both to retailers, and directly to customers through web based mail order.

Currently, SD is experiencing a structural change in the book business where there is a greater flow of goods going directly to the customer instead of through bookstores. This is why the company must adapt to a more direct business approach. SD provides various related services to both retailers and publishers (hereafter called clients). One increasingly popular such service is e-commerce fulfilment. Essentially, it means that the client can start a web shop on the Internet and sell their goods directly to the customer. SD provides the web shop foundation, as well as order fulfilment. This includes stock keeping, delivery and accounting.

The current goal for SD is to expand this service by increasing quality, maintainability and the number of clients using it.

1.2 Purpose

The purpose of this thesis is to develop an interface for SD that enables clients to create and maintain a web shop online.

1.3 Problem description

A successful system needs to 1. meet the demands of the clients, and 2. be designed in a way so that the clients are willing to use it. Thus, the first step towards a solution involves understanding what essential properties the proposed system should have, based on clients' needs and skills. Second, we need to be assured that the final system reaches a certain level of usability.

1.4 Solution method

This thesis will focus on methods for requirements engineering to ascertain the desired properties of the system, while keeping a usability-centred approach for the design of the user interface.

1.5 Scope

The focus for this report is on principles for Usability Engineering and Requirements Engineering. Only details required for a brief understanding of the project is supplied concerning database technology and the web platform ASP.NET. The report does not describe in any detail the technical design, implementation, or testing of the final system.

1.6 Sources

Most of the references are to well-established authors on their subject. The books on ASP.NET – Anderson et al and Conrad et al – seemed to have a rather positive bias towards ASP.NET (like making it appear superior to other technologies), so I had to be careful to extract facts from opinion in these two sources. Where these hard copies were not sufficient, I resorted to Microsofts on-line sources for latest version facts of ASP.NET. These sites are strictly commercial, so selection of plain facts was necessary, and some references in this report may be subject to bias.

1.7 Structure

1.7.1 Outline

Chapter 2 is recommended as it gives further insight into the problem and the intended solution. Chapter 3 describes the foundation of Software Engineering and Usability Engineering, of which methods have been applied in this report. Chapter 4 covers some technical aspects of ASP.NET and database design. This chapter may be skipped if the reader is already familiar with these subjects. The most interesting chapters in this report are chapters 5 and 6, which describe the requirements gathering and the development of the user interface. Chapter 7 is also a major chapter, which describes the final system design and implementation. A discussion of the results follows in chapter 8.

1.7.2 Abbreviations

ADSL	Asymmetric Digital Subscriber Line
ASP	Active Server Pages
CSS	Cascading Style Sheets
HCI	Human-Computer Interaction
HTML	Hypertext Markup Language
MI2	Merchant Interface 2.0
PS2	Partner Shop 2.0
SA	Super Administrator
SD	Star Distribution (StjärnDistribution AB)
SQL	Structured Query Language
UI	User Interface
UML	Unified Modelling Language
WYSIWYG	What You See Is What You Get
XML	Extensible Markup Language

1.7.3 Glossary

Client	A stakeholder and user of the MI2. Is usually a publisher.

Data Access Layer	An interface to the database that connects to the database and supports retrieving and updating datasets.
GridView	A server control in ASP.NET that displays data in a tabular format, and supports modification of data.
Merchant Interface	The system to be developed as an interface between the client and the raw data storage.
Repeater	A server control in ASP.NET that displays data in a format specified by templates.
TreeView	A server control in ASP.NET that displays hierarchical data in a tree structure.
ViewState	A feature in ASP.NET that persists data between requests, in the otherwise stateless web components.

2 Problem description

This chapter will give more detail to the problem and the intended solution.

2.1 Overview

The system we are going to look at is a web shop foundation. It was created as a service to publishers (or clients) that are connected to and sell their products through Star Distribution. The system lets a client create a web shop for the end customers, without any knowledge required in programming or web design. The content of the web shop is customizable, but follows a basic template.

2.2 Existing system

The existing system has two major parts; Partner Shop 1.0 (the front-end), which is the web interface to the *customer*, and the back-end where the data is managed by the *client*. Currently, clients have to work through staff at Star Distribution, which I will refer to as *super administrators* (SA). See *Figure 2.1*.

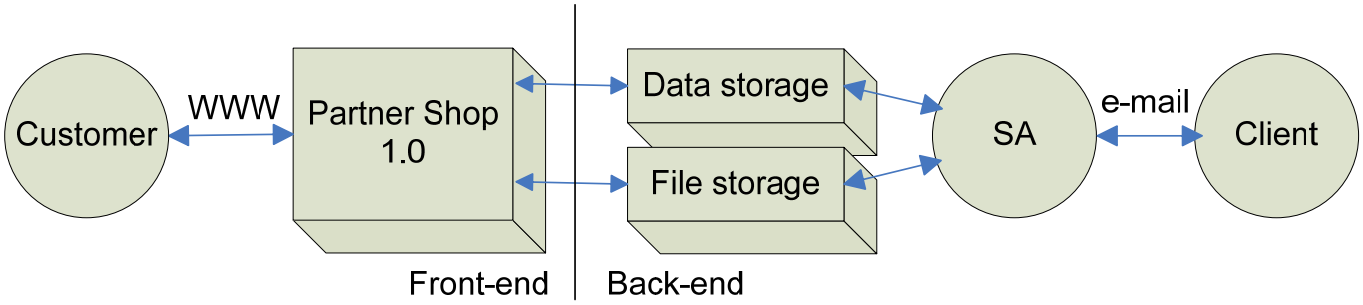

Figure 2.1: Overview – existing system

2.2.1 The web shop's front-end

The web shop front-end is the web interface that the customers use. See *Figure 2.2* for an example of a web shop built on the existing system. To the left is the navigation menu, starting with category pages, a search box, the shopping cart and checkout, and lastly help and other information. The main frame contains a product listing.

Figure 2.2: Sample web shop of existing system

This interface has most of the features you could normally expect from a web shop:

- A customized front page, with news and offers.
- Customized general information pages.
- Product categories organized in a tree structure.
- A product search feature.
- A product list displaying images, and brief information.
- A product details page, displaying a larger image and full information, as well as related products and products by the same author.
- A shopping basket and a checkout page.

The data behind this front-end is what the client is supposed to manage in the back-end (or merchant interface).

2.2.2 The web shop's back-end

The e-commerce system was originally designed with focus on functionality, not usability. It is written in ASP on top of an SQL-server database. There are several limitations in this system. Products can only be imported by sending a spreadsheet to the staff through email.

The client needs to be able to customize the web shop for example by editing layout, styles, contents and shop settings. This dynamic content is stored in a database. Today there is no intended user interface to this database, and consequently the only way to modify the information is by manually editing table data. *Figure 2.3* gives you an idea of how the direct database interface looks like. Modifications can only be made by SAs, and thus makes it complicated and cumbersome for the client to maintain the system. This is because the client has to rely on the SA for modifications, and the fact that no immediate effect of the modifications can be seen until maybe the day after.

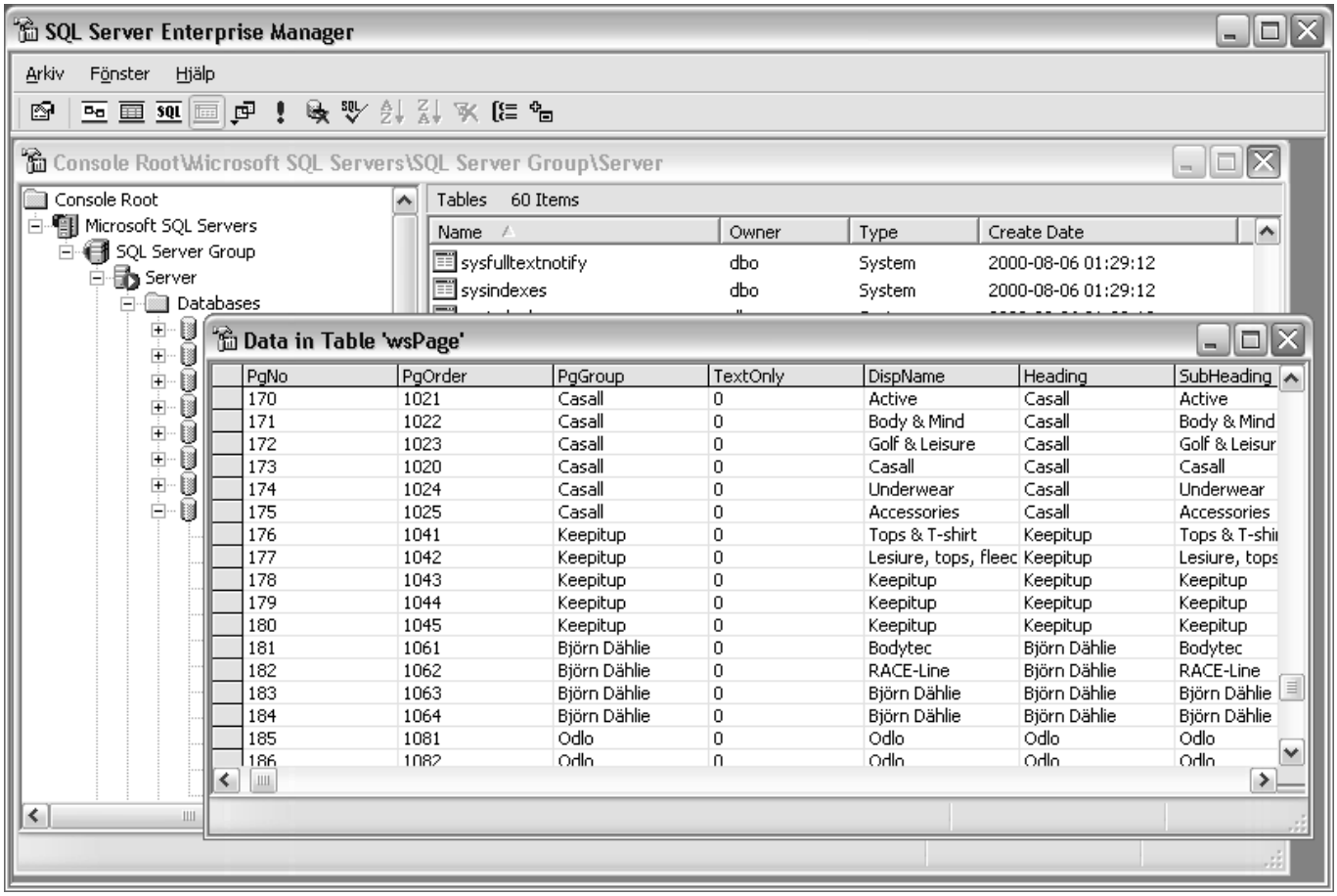

Figure 2.3: Back-end of existing system

2.3 Suggested solution

A solution has been suggested where the client should be able to do all the editing himself using an online interface, referenced to as *Merchant Interface 2.0* (MI2). See *Figure 2.4*.

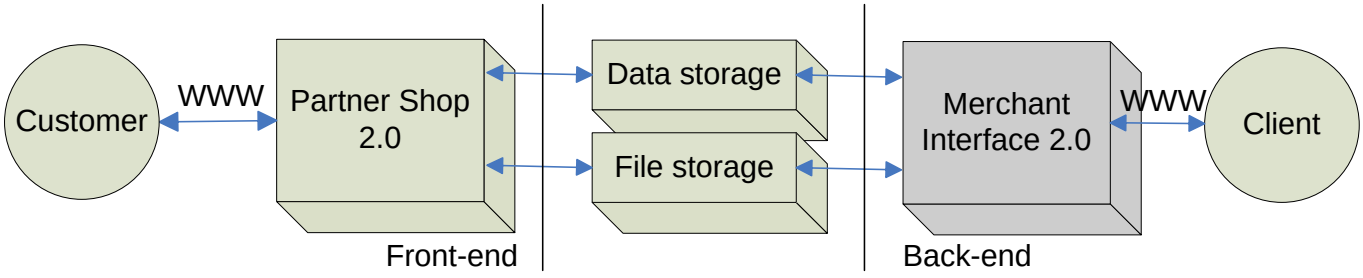

Figure 2.4: Overview – suggested system

Planning for the upgrade, the developers feel a need to deviate from the use of ASP due to the difficulty in maintaining code and code libraries. Instead, ASP.NET 2.0 (see chapter 4) has been suggested as the main platform for MI2.

The idea for the new system has come naturally, as the demand for the service has increased, and Star Distribution has received requests from clients as time has passed.

The main features of the suggested system are:

- The client can perform all editing online.
- Products can be selected from Star Distribution's database and added to the shop.
- Images can be uploaded and used for products and other pages.
- Creation of categories and product attributes.
- Creation of campaigns and special prices.
- Creation of customer accounts, including retailer accounts with a separate login.
- Possibility of localization support.

The requirements for the new system are mostly based on the existing system and suggested improvements from clients.

Creating an interface for the data and file storage (such as the MI2) would not necessarily affect the front-end. Nevertheless, the new set of requirements also concerns the front-end, whereby this system also requires a redesign. However, the front-end is outside the scope of this thesis.

3 Methodology prerequisites

This chapter focuses on the methodologies, which acts as a base for the development process of the system. The reader that is familiar with requirements engineering, design and testing may find the contents of section 3.1 tedious. Similarly, the reader that is well acquainted with usability might want to skip or skim through section 3.2.

3.1 Software Engineering

This section explains some fundamental methodologies of Software Engineering that are used in the development of the MI2.

3.1.1 Requirements Engineering

Capturing requirements is aimed at understanding what the customers and users expect the system to do. The process is called *Requirements Engineering*.

> A requirements engineering process is a structured set of activities which are followed to derive, validate and maintain a systems requirements document. [2, p. 41]

The Requirements Engineering process typically consists of four activities; gathering requirements, understanding and analyzing them, documenting or representing them, and finally validating them [2, p. 49]. The activities may be interleaved, and thus may not necessarily be performed in a strict order. They are outlined here:

1. *Elicitation.* This task is aimed at understanding what the customer really wants. Useful techniques include examining previous systems, looking at existing documents, questionnaire surveys, workshops, scenario-based techniques, interviews etc. In addition, we might have reusable requirements from reuse libraries or suggested types of requirements from templates [7, p. 144].
2. *Analysis & Negotiation*. Once elicited, requirements need to be thoroughly analyzed for conflicts and inconsistencies to avoid problems further ahead in the development. Feasibility checking might involve prototyping. Requirements are also usually categorized and prioritized. See section below for additional information.
3. *Documentation*. To avoid ambiguity and stick to a set of quality attributes described below, often a more formal expression method than natural language is used in this process. Many techniques have been suggested, including data-flow models, state-transition diagrams, object oriented models and entity-relationship models.
4. *Validation.* At last, the requirements document needs to be validated, both by the stakeholders so that it describes what they need, and by the system designers so that it is sufficient for the design.

Requirements elicitation makes it possible to write a *requirements definition* document. The requirements definition is a complete listing of what the customer expects the system to do, in terms that the customer can understand [7]. The document serves as an understanding between the developer and the customer.

A *requirements specification* document describes the requirements definition in technical terms, suitable for the development of a system. Thus, care must be taken so that no information is lost or changed during the transformation [7].

If the customer has good technical skills, the requirements definition and requirements specification can be combined into one document.

Characteristics of requirements

Whether it is documented, implicit, or is yet to be made aware of , “A requirement is a feature of the system or a description of something the system is capable of doing in order to fulfil the system’s purpose” [7, p. 136]. There are a number of desirable quality properties for each requirement in the specification. Some of them are outlined here [7, p. 145][2, p. 46]:

1. *Correct*. Both developers and customers need to make sure that the requirements are without errors.
2. *Consistent*. Requirements need to be stated without chance of ambiguity or inconsistency.
3. *Abstract*. A requirement should be implementation independent, i.e. it should state “what” rather than “how”.
4. *Realistic*. Requirements must be possible to fulfil.
5. *Needed by the customer*. If a requirement does not work directly to solve the problem, it might incur unnecessary restraints on the developers.
6. *Verifiable*. It must be possible to write tests to assure the customer that the requirement has been met.
7. *Traceable*. Each system function must be traceable to the requirement that mandated it.

Functional and Non-functional requirements

We can divide requirements into two major groups. Functional requirements tell what a system does, as opposed to non-functional requirements that define how a system operates. More specifically, functional requirements describe interactions between the system and its environment, or the systems reactions to certain stimuli [7, p. 141].

Non-functional requirements focus on every aspect of the system *other* than the functional. They restrict our choices in constructing the system.

As an example, a functional requirement might describe that a certain system must display a report at the click of a button. If it takes ten minutes for the report to be displayed, it is still fulfilled by that requirement, but it might make the user loose patience. A non-functional performance requirement could remedy the problem by stating that a report must be displayed within five seconds from the initial request.

Types of requirements

Many types of requirements can be included in the requirements definition document, including these, as suggested by Pfleeger [7, p. 142]:

- *Physical Environment*. Environment, locations and environmental restrictions.
- *Interfaces*. Input and output interfaces, formatting, medium.
- *Users and Human Factors*. Focuses on whom the users are and if there are several types of users, skill level, required training and usability issues.
- *Functionality*. Describes what the system does and when, modes of operation and performance constraints.
- *Documentation*. Type and amount of documentation and for which audience.
- *Data*. Defines data format, timing and accuracy as well as data storage.
- *Resources*. Required material and human resources, developer skills. Prescribed timetable and budget.
- *Security*. Access control, backup, data and program isolation.
- *Quality Assurance*. Requirements for reliability, availability, maintainability, security, portability etc.

Prototyping requirements

If we are not sure that the requirements document is complete or realistic, we may want to investigate this through a prototyping activity. Pfleeger [7, p. 169] describes two different approaches, known as *rapid prototyping*: A *throw-away prototype* is intended to explore a problem or feasibility of a solution, but is not intended to be used as a part of the final design. On the other hand, an *evolutionary prototype* is developed to learn more about the problem, and if the customer approves, it will be used as a basis for the actual interface.

3.1.2 Design

Once the requirements documents are done, we can create a design that will satisfy all the needs.

> **Design** is the creative process of transforming the problem into a solution; the description of a solution is also called **design**. [7, p. 195].

Usually the design is described in two documents. The *conceptual design* describes to the customer what the system will do in a language that the customer can understand. Once the customer approves of the conceptual design, it is translated into a *technical design* that details the system to the developers.

The design process is iterative. After reviewing the conceptual design, the customer might change the specifications as he reacts to what he likes or dislikes. In addition, circumstances could change the customer's needs.

When the system has been designed, it should be checked against the customer's requirements to ensure they are satisfied. This is called *validating* the design. Thereafter, a

quality control known as *verification* is performed; making sure the design adheres to good characteristics of design.

Decomposition and modularity

Designing the system involves starting with a high-level description and successively working through lower levels as level of detail is increased.

> A design can be derived by working from system data descriptions, events, user inputs, high-level function descriptions, and creating a hierarchy of information with increasing detail [7, p. 199].

In particular, the design includes information on how a system is to be implemented, as opposed to the requirements specification. The decomposition methods separate the design into composite parts, called *modules* or *components*. Modularity is a good system characteristic, where the system is composed of well-defined components. “A component is well-defined if all inputs to it are essential to its function, and all outputs are produced by one of its actions” [7, p. 200].

Architecture

Pfleeger [7, p. 201] suggests design on three levels; architecture, code design and executable design, but points out that the process seldom is linear. Instead, designers move back and fourth between the stages, as more understanding is gained by attempting to solve the problem.

- *Architecture*. System capabilities from the requirements specification are associated with the components that make up the system, and interconnections are defined between them.
- *Code design*. Here, components are programming language primitives, tied together by primitive operators such as arithmetic operators and data manipulation routines. Composition is possible by arrays, files and procedures.
- *Executable design*. Details the code design at the level of memory allocation and data format etc.

Many architectural design methods have been proposed, including pipes and filters, object-oriented design, implicit invocation, layering, repositories, interpreters and process control [7].

General structures for software design have been developed, called *design patterns*. Commonly referenced is Gamma, Helm, Johnson, Vlissides [5], who describes design patterns as “descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context”.

Prototyping

If uncertainty arises, whether a feature is feasible or not, it might help to develop a prototype. In this way, many problems can be resolved before coding begins. A prototype usually focuses on particular aspects of the system, and omits performance details and

some functionality. As with requirements prototyping, exploring feasibility with low fidelity and thereafter discarding the prototype is called *throw-away prototyping*.

> Brooks (1975) recommends building a system, throwing it away, and building it again, so that the second system will profit from our learning as we discover mistakes made in the process of building the first [7, p. 236].

Studies show that development processes using prototypes produce results comparable to those of traditional techniques, but with 40 percent fewer lines of code and 45 percent less effort [7, p. 236].

3.1.3 Testing

Testing is an activity focused on finding faults. It is essential to ensure quality. A fault usually means that the system does not work as described in the requirements.

There are several possible causes of a failure [7, p. 332]:

1. The specification may be wrong or have a missing requirement.
2. The specification may contain a requirement that is impossible to implement, given the prescribed hardware and software.
3. The system design may contain a fault.
4. The program design may contain a fault.
5. The program code may be wrong.

Types of faults

In order to be able to write exhaustive tests, it is important to be aware of the possible types of faults that can occur [7, p. 333]:

- *Syntax fault* is an error in use of the constructs of the programming language. Many faults are detected by compilers.
- An *algorithmic fault* occurs when a component does not produce the expected output given a certain input, due to problems with the processing steps.
- *Computation* and *precision faults* arise when a formula is wrong or the result has a bad accuracy.
- A *documentation fault* is apparent if the documentation does not match what the program actually does.
- *Stress* or *overload faults* occur when data structures with multiple items are filled beyond their specified capacity.
- *Capacity* or *boundary faults* appear when performance degrades unacceptably as the systems capacity reaches its specified limit.
- *Timing* and *coordination faults* occur in real-time systems when events are badly coordinated.

- *Throughput* or *performance faults* occur when the system does not perform at the specified rate.
- *Recovery faults* occur when a failure takes place but the system does not behave as desired.
- *Hardware and system software faults* can arise when the specified hardware and software do not work according to the specification.

Testing steps

Testing is usually conducted in several steps, especially for large systems. Focus varies from individual units to the whole system and from isolated tests to tests in the specified environment [7, p. 337].

1. *Unit test*. Each program component is tested on its own. For a given set of inputs, output is compared to the expected output. In addition, code reviews such as code inspections and code walkthroughs are powerful methods for discovering faults. Some platforms support writing automated unit tests.
2. *Integration test*. The components are tested together to verify that their interfaces are defined and working properly as described in the design specification.
3. *Function test*. This test ensures that the system functions as described in the requirements specification.
4. *Performance test*. The system is tested according to hardware and software requirements in the working environment.
5. *Acceptance test*. The system is checked against the customer's requirements description with the customer present.
6. *Installation test*. The system is installed and verified in the environment where it will be used.

3.2 Usability Engineering

This section will elaborate on the methodology used for the user interface design process in the MI2 project, particularly focusing on Usability Engineering.

3.2.1 Introduction

Human-computer interaction (HCI) is a study of interaction between humans and computers [2, p. 22]. It is an interdisciplinary subject, combining computer science with other areas of study and research.

Design methods within HCI usually involve the concept of *usability*. The International Standards Organization defines usability in the following way:

> [Usability is the] extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use [2].

If we break it down, we see that *specified users* – not any type of user – are being focused. Achieving *specified goals* in a *specified context of use*, means that goals are set in advance, as well as the context of use. *Effectiveness, efficiency and satisfaction* are what we need to measure to see if the goals have been met.

Notably, usability is not a typical property of the system as most attributes are described in a requirements specification, but rather a quality of the interaction between the user and the system. The extent to which users participate in the development is determined by the choice of design process method.

Theory-based approaches rely on general UI design guidelines, and on established general characteristics of users interacting with computers. Here users do not participate at all. However, according to Carlshamre [2, p. 27], these methods have only had limited success.

On the other end, *Participatory Design* calls for full user involvement, and thus requires training and active cooperation. It can bring accurate information about users' tasks and needs. However, it might be costly and time-consuming, and care need to be taken not to upset people who are not involved or whose suggestions are rejected [8, p. 125].

Usability Engineering focuses not on user participation or on maximizing usability, but on reaching a specified level of usability. Here, users may be involved in requirements elicitation and usability testing. The recommended activities for Usability Engineering are as described by Carlshamre [2, p. 30]:

1. User profiling
2. Task analysis
3. Defining usability goals through metrics and setting desired levels of usability to be achieved
4. Analyzing the impact of possible design solutions (redesign)
5. Incorporating user-derived feedback in product design (prototype)
6. Evaluating the design by usability testing and iterating the development until the planned usability levels are achieved

The evaluation may lead to a redesign, and possibly to a change in usability goals and specification. The process is illustrated in Figure 3.1. The first few steps are described in the following sections.

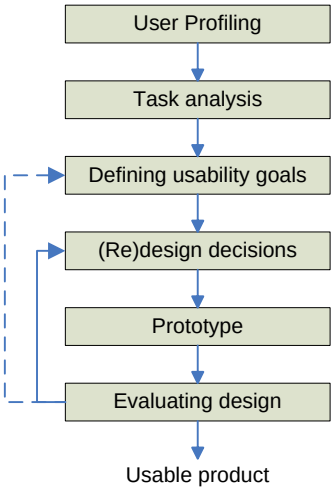

Figure 3.1: Usability Engineering activities

3.2.2 User profiling

It is important to recognise the diversity of the intended users. Designers need to be aware that users learn, think and work in different ways, as Shneiderman & Plaisant [8, p. 67] put it:

> Every step in understanding the users and in recognizing them as individuals with outlooks different from the designer's own is likely to be a step closer to a successful design.

Therefore, we need to understand both the community of users, and their skill levels. Popular methods for acquiring data include questionnaires, or actually testing users' skills and knowledge in general or in a domain of concepts.

This job will result in a user profile, which the designers of the user interface will have to respect. Novice users will often want more feedback and simplicity, while experienced users usually prefer less distracting feedback and denser layout [8, p. 69]. It naturally gets more complicated when different usage classes have to be accommodated. Even if this profile will guide the design in some ways, "… most of the interpretation is left to the experience and skill of the designer" [2, p. 31].

3.2.3 Task analysis

For the system to support the "right" tasks, developers must first identify and understand the tasks to be performed. The result is a document that describes what tasks the users should be able to perform with the system.

Developers must know both the task sequences, and which tasks are common. Frequent tasks need much attention to make them quick and simple, even at the expense of lengthening some infrequent tasks[1] [8, p. 69]. In the end, they must decide what tasks to actually support, since an interface easily can get cluttered and confusing with too many options.

[1] Consider for example special keys such as "Tab" and function keys in comparison to the lengthier menu selection.

The typical way of capturing this knowledge is by observing users in their work environment, by interviews and surveys, and by studying work descriptions.

3.2.4 Usability metrics

To be able to evaluate our design according to specification, we need a way to measure usability, in the form of goals and usability metrics. Carlshamre [2, p. 34] interprets the ISO definition of usability in the following way:

> … to verify usability we have to study the specified users attempting to solve the specified tasks in the specified context, and see if they can do this according to our own specified standards.

Thus, data is obtained during testing from a carefully constructed set of benchmark tests, in a determined environment.

Shneiderman & Plaisant [8, p. 16] suggests the following usability measures:

1. Time to learn
2. Speed of performance
3. Rate of errors by users
4. Retention over time
5. Subjective satisfaction

A comprehensive list of different usability measurement criteria according to the attributes relevance, efficiency, learnability and attitude was given by Carlshamre [2, p. 35].

3.2.5 Evaluating Interface Designs

There are two main methods of evaluating an interface design; expert reviews and usability testing.

Expert reviews

Formal reviews done by experts in HCI or the application domain have proven to be very effective in identifying usability problems.

> It has been suggested that by (a) heuristic evaluation 75% of all usability problems could be discovered, if carried out by 5 different evaluators. However, the method requires that the evaluators have extensive experience to be effective. [2, p. 37]

Shneiderman & Plaisant [8, p. 142] lists the following five expert-review methods.

- *Heuristic evaluation*. Experts review an interface and check conformance against a list of design heuristics, such as the eight golden rules included in *Appendix A – Questionnaires*. Requires several experienced HCI-experts to be effective.
- *Guidelines review*. The interface is checked for conformance with the organizational or other guidelines document.

- *Consistency inspection*. Consistency is verified across a family of interfaces, checking terminology, graphical attributes, data formats etc.
- *Cognitive walkthrough*. The experts simulate users doing typical tasks, focusing mainly on frequent tasks, but also checking error recovery and such rare tasks.
- *Formal usability inspection*. A formal inspection meeting is held to present and discuss the interface. Uses many resources.

Expert reviews can be scheduled both early and late in the development process. The report can preferably rank recommendations by importance and expected effort level, to aid in selecting items for redesign.

Usability testing

During usability testing, the actual users are called for to participate. Usability testing is typically performed in usability labs, where users are isolated and videotaped. Other methods perform testing in the specified context or even remotely.

Users are often encouraged to "think aloud" during testing, which usually yields interesting clues to what is going on in their minds, and hints for design flaws.

Both during a "thinking aloud" session and after a test, users often have plenty of valuable comments and suggestions.

Shneiderman & Plaisant [8, p. 148] describes several forms of usability testing:

- *Paper mockups*. Paper mockups of screen layouts are useful in the early stages of design to get response to wording, layout and sequencing. A test administrator simulates a screen by flipping pages, while the user performs common tasks. This type of testing is quick, low cost and usually productive.
- *Discount usability testing*. This method is useful for task analysis, prototype development and testing. Only tree to six test participants are recommended, since most problems are found with only a few participants. The method is especially useful for guiding redesign, since it is rapid.
- *Competitive usability testing*. This approach compares a new interface to a previous version or a competing product. Designers of the test must be careful to construct parallel sets of tasks.
- *Universal usability testing*. This technique attempts to eliminate problems in various configurations of hardware, software platforms and networks and with diverse users.
- *Field tests and portable labs*. This method puts the interface to test in the field, usually logging commands and productivity. A common variant is the wider distribution of a beta release where users may be asked for their comments afterwards.
- *Remote usability testing*. Web based applications makes it possible to have usability tests online, reaching a broad community at a low cost. However, there is less possibility of observing users' behaviour.

- *Can-you-break-this tests*. Here, users are asked to try to find fatal flaws in the system that would otherwise cause significant economic loss and damage to the company to repair after distribution.

Surveys

Written user surveys acts as a complement to expert reviews and usability testing methods. The method is well known and useful in collecting large amounts of statistical data.

> The key to successful surveys are clear goals in advance and development of focused items that help attain those goals [8, p. 150].

Survey forms should first be reviewed and tested by sample users before being distributed.

A commonly used survey instrument to assess users' satisfaction with specific aspects of the interface is the *Questionnaire for User Interaction Satisfaction* (QUIS) developed at the University of Maryland [12].

4 Technical prerequisites

This chapter focuses on the foundation technologies on which the system is based upon. An understanding of Microsoft.NET is useful in preparation of the discussion in chapter 8. The reader who is familiar with ASP.NET and database design may want to skip or skim through sections 4.1 and 4.2 respectively.

ASP.NET was chosen as the web platform for the new MI2 system. The previously used classic ASP technology had made code difficult to maintain since imperative programming statements were mixed with declarative HTML statements. SD was already using a family of Microsoft products for their servers, and a recent upgrade to the system had included ASP.NET 2.0. This, as well as existing knowledge with ASP made it natural to advance to ASP.NET.

The proposed system relies heavily on a database, which makes it relevant to look at database design, although this report will focus very little on this aspect. Again, the existing database software is Microsoft SQL Server 2000. This has worked very well in the past, and there is no apparent reason to switch to an alternate platform.

4.1 Microsoft .NET

The first section below gives an overview of the .NET Framework and the technologies involved. The second section digs deeper into some interesting aspects of ASP.NET. Concepts from these two sections are used throughout the report, and are further discussed at the end of the report.

At the time of writing, the current version of the .NET Framework is 2.0. Still, most of the hard copy references I found were for version 1.0, which is why some technology references are from Microsoft’s web sites.

4.1.1 .NET Framework

The .NET Framework is a platform for building, deploying and running Web Services and applications. It covers the operating system so that applications are insulated from specific operating system details such as memory management, file handling etc.

The .NET Framework consists of four major parts, as shown in *Figure 4.1* [9]. Each part makes use of one or more of the lower layers.

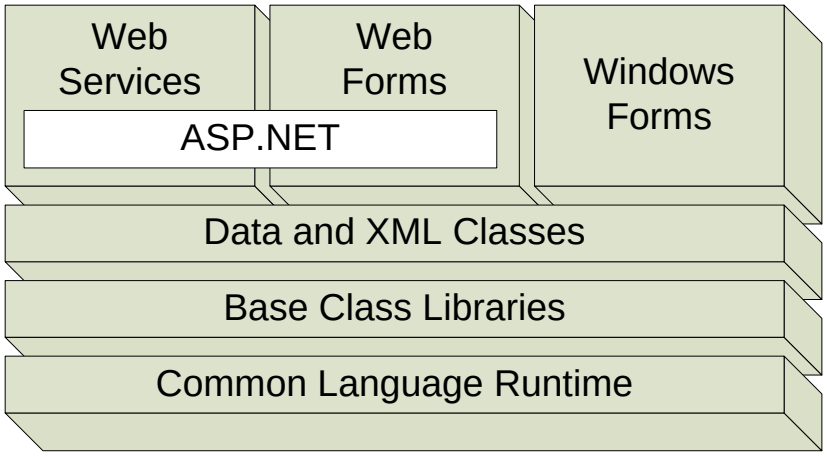

Figure 4.1: Microsoft .NET Framework

The parts of the .NET Framework are outlined in the following sections.

Application development technologies

The top layer includes user and program interfaces:

- *Windows Forms* is a technology for creating standard desktop applications.
- *ASP.NET* is a technology for building web applications, which includes *Web Services* and *Web Forms*.
- *Web Services* are applications based on a standardized XML protocol called the *Simple Object Access Protocol* (SOAP). It allows applications to exchange data over the HTTP protocol, yet shielding consumers from implementation details of the server platform.
- *Web Forms* provides a forms-based user interfaces for web applications. It allows for declarative, component-based programming with server-side code for event handling and other methods.

Data and XML Classes

These high-level classes support data access and XML. The ADO.NET enables data access through the OLE DB, ODBC, Oracle, and SQL Server interfaces. XML classes support XML manipulation, searching and translations [9].

Base Class Libraries

The Base Class Libraries (BCL) provides the fundamental building blocks of all applications and serves as the main point of interaction with the CLR. Classes are as usually organized into namespaces, covering areas such as Collections, Thread Support, Code Generation, IO and Security [1, p. 63].

The Common Language Runtime

The *Common Language Runtime* (CLR) is the execution engine for all .NET Framework applications. Code written for the .NET platform requiring the CLR to function is known as *managed code*. Applications are first compiled into a CPU-independent common language, namely the *Microsoft Intermediate Language* (MSIL). This allows for cross-

language integration. Prior to execution, MSIL is converted into native machine code by the CLR, a process known as *just-in-time* (JIT) *compilation* [1, p. 43].

Important to the interoperability and implementation of multiple programming languages is the *Common Type System* (CTS) that supports types and operations found in most programming languages.

Some of the major services of the CLR are listed here [3, p. 35]:

- *Cross-language integration*. Managed code written in one language can be integrated with code written in other languages, including cross-language inheritance and exception handling.
- *Code verification*. Code verification enforces type safety, which prevents code from performing illegal operations. Instead of crashing, the CLR throws an exception.
- *Security enforcement*. Managed code declares a required security permission level, and the CLR is responsible for verifying if permission is granted or not depending on the security configuration of the machine.
- *Memory management*. The CLR is responsible for application memory isolation and garbage collection (GC)[2].

4.1.2 ASP.NET

ASP.NET is a set of technologies for creating web application and XML Web Services. ASP.NET pages execute on the server and generate markup in languages such as HTML and XML. Page and Web Service logic can be written in a wide range of languages for the .NET Framework[3].

Programming model

The programming model is *event-driven*, which enables the separation of application logic and user interface. The server-side logic can be placed in so-called *code-behind* files, making the main page free of code other than purely declarative statements. As illustrated in *Figure 4.2* below, the web form and code file are tightly coupled. In this example, the code-behind file is written in C#. The declarative statements of the web form are automatically transformed into programming statements, which lets the programmer refer to the objects from code. Furthermore, the web form actually inherits from the resulting class, which then gives the page access to all the protected objects.

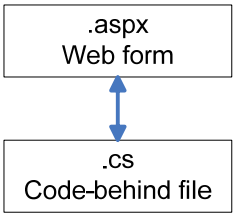

Figure 4.2: Web forms and code behind files

[2] As a side effect of the nondeterministic nature of the GC, no dependable code should be put in class destructors.

[3] Microsoft has announced the existence of over 20 compilers for the .NET Framework [9].

The whole page is compiled, including all of the HTML and other text, to a class. The class can then be executed to create output for the browser. Each server component is declared with an attribute `runat="server"`, which makes them objects within the page class. These objects expose properties, and make it possible to call methods and respond to events raised on the server [1, p. 187].

Key features

This section summarizes some of the key features of ASP.NET.

- *Declarative programming*. ASP.NET server controls enable declarative programming similar to HTML with little coding compared to classic ASP [10]. The programming model is also cleaner since application logic is separated as discussed above.
- *Performance*. Performance can be increased with caching on several levels, such as output caching and database caching. The fact that classes are compiled instead of interpreted also play a part in performance.
- *Standards based*. Web controls support rendering valid, well-formed XHTML markup, designed to work in all browsers.
- *Design aid*. Consistent layout is supported by the use of Master Pages, which acts as page templates. Consistent appearance is simplified with page-wide Themes that are based on style-sheets.
- *Security and protection*. ASP.NET membership services offer support for managing access to web applications. ASP.NET provides built-in protection from common forms of hacker attacks, including cross-site scripting and request-replay attacks [10].
- *Data-driven programming*. Data presentation is managed through a set of server controls that are designed to work with data from various types of sources, by means of a process known as *data binding* [1, p. 283]. Repeated-value data binding abstracts the details of fetching data records and formatting output, making data presentation a purely declarative task.

Table adapters

There are many different types of data sources in ASP.NET. One of them is the TableAdapter. Table adapters provide communication between the application and the database. Table adapters are classes created by the dataset designer tool in Visual Studio 2005. A table adapter object can connect to the database, execute queries or stored procedures, and return a typed data table.

Repeated value data binding

ASP.NET provides a set of controls for repeated data binding. They expose properties that allow them to be bound to different types of data sources, and then automatically display one row or item for each item in the dataset. Some controls also support the updating or

insertion of data. Examples of data sources include the `ASP:SQLDataSource`, `ASP:XMLDataSource` and `ASP:ObjectDataSource` components.

Here is a selection of some controls designed for repeated data binding [1, p. 283]:

- The `ASP:DropDownList` control creates a HTML `<select>` list with one `<option>` element for each data item.
- The `ASP:CheckBoxList` control creates a list of HTML input checkboxes, one for each data item.
- The `ASP:Repeater` control repeats the contents specified by a template, once for each data item. No additional HTML formatting is applied by the control.
- The `ASP:DataList` control repeats the contents specified by a template, once for each data item. Items can be repeated either as a list, or be placed in a `<table>` and repeated both vertically and horizontally. The `ASP:DataList` supports the selection, editing and deletion of items.

The following controls are new to .NET Framework 2.0 [11]:

- The `ASP:GridView` control displays data in a tabular format, and automatically creates columns for each field in the data set. It supports paging and sorting, as well as the selection, editing and deletion, but not the insertion of items.
- The `ASP:DetailsView` control displays the values of a single record from a data source in a table, where each displayed row corresponds to a field of the record. The DetailsView control supports editing, deleting and inserting records.
- The `ASP:FormView` control is similar to the DetailsView control, except that data is displayed using user-defined templates.
- The `ASP:TreeView` control displays hierarchical data, such as a table of contents, in a tree structure.

4.2 Database design

This section focuses on methods for data modelling and explores some considerations for database design.

4.2.1 Introduction

Users interact with a database usually through one or more computer programs, referred to as a database application. *Figure 4.3* shows the components of a database system. The database application manipulates data through the *Database Management System* (DBMS), using a *Database Manipulation Language* (DML) such as the *Structured Query Language* (SQL). To create and manipulate the structure of the database, an administrator would typically work through a *Database Definition Language* (DDL) against the DBMS. Again, SQL is an example of this. The DBMS serves as an interface to the database and manages the physical structure and data on disk.

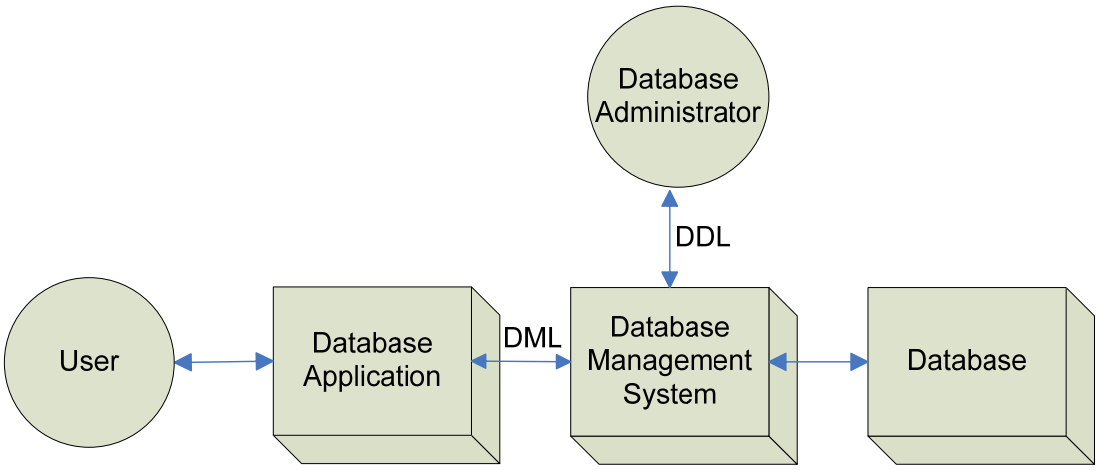

Figure 4.3: Components of a database system

4.2.2 Data modelling

The three schema model was first published 1975 by the American National Standards Institute/Standards Planning and Requirements Committee (ANSI/SPARC), and provides perspective on the role of data modelling and database development [6, p. 32]:

1. An *external schema* is a representation of the way a user views the database. There can be several external schemas for a database, which normally only shows a part of the database.
2. The *conceptual schema* is a complete logical view of the database, which describes all data and relationships in the database.
3. The *internal schema* is a representation of the conceptual schema as it is stored physically on disk.

Building a database begins with constructing a conceptual schema, that will support the desired external schema, but is independent of the internal schema.

The *Entity-Relationship* (E-R) *model*, first published by Peter Chen in 1976 [6, p. 35], can be used to create conceptual schemas. The *extended E-R model* later added subtypes to the E-R model. Another version, the *Integrated Definition 1, Extended* (IDEF1X), became popular as it was standardized in 1993 by the National Institute of Standards and Technology. IDEF1X is based on the same ideas as the E-R model, but uses a different set of symbols. The Unified Modelling Language (UML) presented the E-R model with its own set of symbols and an object-oriented approach [6, p. 36].

Common for all the E-R modelling versions are entities, attributes and relationships [6, p. 37]:

- *Entities*. An entity corresponds to a “thing” in the real world, whether it has a physical or conceptual existence. Entities of the same type are grouped into *entity classes*.
- *Attributes*. Entities have attributes that describe their characteristics. An entity *instance* will have a value for each attribute.

- *Relationships*. Entities can be associated with each other by relationships, which also may have attributes. Relations can involve two or more entity classes, but a degree of two (binary) is most common. There are three types of *binary relationships*; 1:1 (one-to-one), 1:N (one-to-many) and N:M (many-to-many).

Other constructs in E-R model variants include supertype / subtype entities[4], weak entities and domains:

- A *subtype entity* represents a special case of another entity, called a *supertype entity* [6, p. 43].
- A *weak entity* in the extended E-R model cannot exist in the database without the presence of a second entity [6, p. 41].
- The IDEF1X model defines a domain as a named set of values that an attribute can have for added specificity [6, p. 55-56]. A *base domain* has a data type and possibly a value list or range. A *type domain* is a subset of another domain.

4.2.3 Relations

A relation is a two-dimensional table with a number of characteristics. Kroenke [6, p. 120] describes these characteristics the following way:

- Rows contain data about an entity
- Columns contain data about attributes of the entity
- Cells of the table hold a single value
- All entries in a column are of the same kind
- Each column has a unique name
- The order of the rows and columns is unimportant
- No two rows may be identical

A table does not necessarily have these properties, but usually in database terms, the word "table" refers to a relation.

4.2.4 Types of keys

"A key is one or more rows of a relation that uniquely identifies a row" [6, p. 122]. A key is either *unique* or *non-unique*, which determines if duplicate keys are allowed or not. If a key is made up of more than one attribute, it is called a *composite key*. If there are many *unique* keys in a relation, they are called *candidate keys*. Any one of the candidate keys can be chosen to be the *primary key* of a relation. The primary key uniquely identifies rows and represents rows in relationships. In addition, many DMBS organize storage for the relation according to the primary key, and make indexes for fast access [6, p. 123].

[4] Supertype / subtype entities are known in IDEF1X as generic / category entities.

4.2.5 Normalization

Depending on their structure, some relations suffer from what is known as *update anomalies*, causing undesirable consequences [6, p. 125]. A *deletion anomaly* occurs when the deletion of a row causes information of two or more entities to be lost. An *insertion anomaly* occurs when the structure of a relation forces the insertion of two or more entities at the same time. A *modification anomaly* occurs when the update of a row forces the update of other rows to avoid inconsistencies [4, p. 472]. Many classes of anomalies can be eliminated by *normalizing* relations, into classes of *normal forms*.

Before looking at these classes, we need to define the term *functional dependency*:

> A *functional dependency* occurs when the value of one attribute (or set of attributes) determines the value of a second attribute (or set of attributes). The attribute on the left side of the functional dependency is called the determinant. [6, p. 144].

Kroenke [6, p. 128-133] defines the normal forms as follows:

- A relation is by definition in the *first normal form*.
- "A relation is in *second normal form* if all its non-key attributes are dependent on all of the key".
- "A relation is in *third normal form* if it is in second normal form and has no transitive dependencies".
- "A relation is in *Boyce-Codd normal form* (BCNF) if every determinant is a candidate key".
- "A relation is in *fourth normal form* if it is in BCNF and has no multi-value dependencies".

There is also a *fifth normal form*, but the implications of the involved dependencies are unknown.

Because no certainty was gained that all problems were eliminated with the above normal forms, the *domain/key normal form* was developed that guarantees that there will be no anomalies:

> A relation is in *domain/key normal form* (DK/NF) if every relation is a logical consequence of the definition of keys and domains [6, p. 133].

A *constraint* is any constraint on the static values of attributes whose truth can be evaluated, and a *domain* is a named set of values that an attribute can have.

However, there may be instances when normalization is not desired. This could be the case if a normalized design would otherwise be unnatural, awkward or result in unacceptable performance [6, p. 141].

4.2.6 Synthesis of relations

The task of synthesizing well-formed relations involves inspecting attributes and their functional dependencies. There are three cases [6, p. 138]:

1. If two attributes functionally determine each other, they have a one-to-one relationship.
2. If one attribute functionally determines the other, but not the other way around, they have a many-to-one relationship.
3. If neither attribute functionally determines the other, they have a many-to-many relationship.

Once the appropriate relationships have been discovered, database design can begin.

4.2.7 Database design

During database design, the entity-relationship data models can be translated into relational database designs. Tables are created from entities and columns from attributes. The identifier of the entity usually becomes the *primary key* of the new table. For each attribute, it must also be decided if it should be required or not, known as NOT NULL constraints. There may be a choice of candidate keys. Due to performance and maintainability reasons, “an ideal primary key is short (so that it takes little storage and is fast to process), numeric (easy to index), and seldom changing” [6, p. 153]. If no candidate key has these properties, a *surrogate key* can be added, such as a 32-bit auto-incrementing integer column. The disadvantage of surrogate keys are that values lack meaning, and that duplicate values may occur during database merging [6, p. 154-155].

Referential integrity constraints

Relationships are represented by placing the primary key of one table into the second table, where it is referred to as the *foreign key*. The foreign key implies a *referential integrity constraint* [6, p. 156]. Thus, if the parent row has any child rows, then updates and deletes to the parent row are prohibited. In addition, insertions and updates in the child row’s foreign key must match one of the values in the parent’s primary key. These constraints are enforced by the DBMS when the corresponding action is performed.

Referential integrity actions

Referential integrity actions can be specified as a more flexible method of handling modifications to a relation. Two particularly common actions concern updates and deletes to a parent row. *Cascading updates* causes the value of the foreign key in the child row to be automatically changed when the parent row’s primary key is updated. *Cascading deletes* causes all related child rows to be deleted when the parent row is deleted.

Using SQL in applications

This section will focus on some specific issues when designing a database in SQL.

Check constraints can be applied to columns of a table, and define limits for column values. Common methods include range checking or checking against an enumerated list [6, p. 233]. In addition, check constraints can compare column values and verify data formatting.

A *trigger* is a stored program that is invoked by the DBMS when an action occurs on a specified table or view. A trigger responds to an update, insert or delete event, and can

occur before, instead of, or after a particular event [6, p. 246]. Common uses for triggers include validity checking, setting default values, updating views, and enforcing referential integrity.

A *stored procedure* is a program that is stored in the database and performs some common action. Stored procedures can be invoked by user programs, and can receive parameters as well as return data. Some advantages of using stored procedures include greater security, reduced network traffic and sharing of code. In addition, stored procedures can be precompiled by the DBMS to improve performance.

An *index* is a data structure that improves performance of the database in most aspects[5], and is automatically created for primary and foreign keys by SQL Server [6, p. 385]. Additional indexes can be created, preferably on columns or combinations of columns that frequently occur in WHERE clauses, and on columns that are used for sorting.

[5] The fact that the index itself must be updated on for example inserts causes some overhead.

5 Capturing requirements

How do we know what features the proposed system needs to have? If we would ignore this issue, we might end up with a system that lacks vital features or functionality, or is so hard to use that only the most dedicated clients will choose to use it. The task is about capturing requirements. A well-known process for this purpose is known as *requirements engineering*, and is described further in chapter 3. The method originates from the Software Engineering methodology, and is not typically associated with Usability Engineering. This chapter focuses on the task of eliciting requirements.

5.1 Requirements gathering

This section describes the initial process of gathering requirements for the system.

5.1.1 Purpose

The purpose of gathering requirements is to produce an accurate and well-defined requirements specification, as well as gain an understanding of what the clients expect the system to do.

5.1.2 Method

The basic requirement was that version 2.0 of the system should not remove any important functionality from the previous version. Above that, we needed to find out what our clients required from the new system. The company itself had some ideas of what the new system should do, but we needed to contact our clients to see what their ideas were. To gather suggestions and inspiration, we made some phone interviews and had meetings to discuss this issue with a few of our clients. The ideas were many and diverse. In fact, we got so many ideas that it was necessary to filter out the most desirable and important requirements to be able to finish the project in due time. We also needed to get to know our target group better, things like environment, previous experiences and skills. This activity, part of Usability Engineering, is known as *user profiling* (see section 3.2.2). A user profile is also often seen in a requirements document in traditional Software Engineering, so in this approach, the two methodologies converge.

One could argue that following the user profiling, a *task analysis* should be done, since this is the next recommended activity in Usability Engineering. I chose however not to include task analysis formally in the development process. The requirements elicitation involved a number of techniques that are similar to those of the task analysis, such as interviews and surveys or studying the existing system. Therefore, I thought I already had a good understanding of the users' tasks through the descriptions in the requirements document.

A questionnaire, named *User Questionnaire*, was created to satisfy the two needs of requirements elicitation and user profiling. It is described below.

5.1.3 User Questionnaire

The purpose of the User Questionnaire is to get a better understanding of the users, and to help filter out appropriate features for the proposed system.

Questionnaire overview

The User Questionnaire is made of three parts, outlined below. See *Appendix A – Questionnaires* for the full questionnaire and the acquired data. I sent the questionnaire to eight of our current clients by email, and got them back within a week.

Part 1: Environment

The first part establishes the characteristics of the computer environment where the system ultimately will be used. The questions concern web browsers, connection speed and screen resolution. The results can be used to form *environmental requirements* on the system (see chapter 3).

Part 2: Computer experience

The second part seeks to determine the skill level of the clients by asking them to rate their ability with some more or less common programs. These programs are relevant in that they represent different ways of interacting with a system, such as word processors, forms, databases and HTML. The results help us create a user profile to guide the design, and additionally help us create suitable requirements for *users and human factors*.

A typical question looks like the following:

> 2.7 Do you feel confident in HTML (language for web pages)?
> (Disagree) 1 2 3 4 5 6 7 (Agree)
> >

Part 3: Merchant Interface 2.0

The third and last part establishes the needs of the clients. The clients where asked to select among the suggested features those that were of most use to them, and were encouraged to add their own ideas and suggestions.

A typical option looks like this:

> 3.3 Usability
> [ ] There are short help texts at every field in the interface
> [ ] There is an on-line manual
> [ ] Other: _____

The results serves as a guide to what features are most desirable, and what features are superfluous. This part helps us design a variety of requirements, including *interfaces, functionality, performance, usability* and *documentation*.

Questionnaire analysis

Now we turn to the task of analysing and making use of the questionnaire results.

Users

As we can see from *Figure 5.1*, the age is mostly within 31 to 50 years, and our clients are equally distributed between men and women. The system must be designed with usability in mind for this audience.

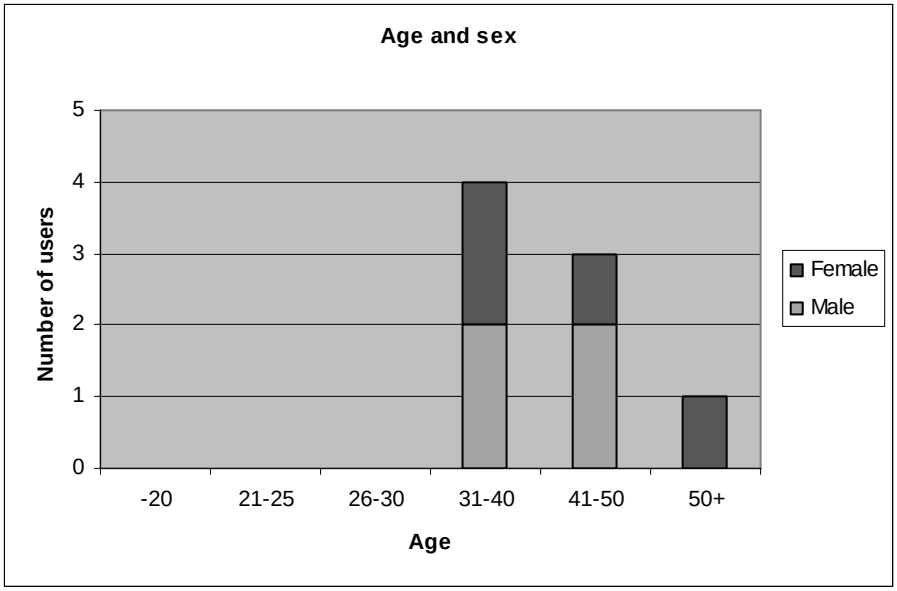

Figure 5.1: Age and sex distribution

Environment

All our clients use Internet Explorer and most have a high-speed internet connection. No conclusion can be drawn about screen resolution, since most did not know this.

This means that we can focus the development for an audience with a medium high bandwidth and foremost on Internet Explorer. Since our target group for MI2 is so small and the users pay to use it, we can simply decide that the minimum resolution of the system is 1024x768 pixels. In the case that any of the clients have problems with this (like having a lower resolution), SD can then provide support on setting this up.

Computer experience

We asked users which computer programs they were familiar with and had used, and then asked them to rate their skill. The results were a bit hard to interpret, since many reported having used a program, but were not at all confident with it.

Six out of seven had used and were familiar with word processors and web editors, while spreadsheets and database programs were not as widely known. Only one felt confident in HTML.

The user ratings are shown in *Figure 5.2*. Users were asked to rate their skill on a scale between one and seven. “1:3” means that tree users rated one on that question. We can se in the figure that many clients reported having quite a bit of computer experience in general, and many still were positive towards learning new programs and systems.

This means that the design can contain some advanced features since users are happy to learn. A design resembling a word processor will probably be accepted by most users. Avoid features and terminology from database and spreadsheet, or explain them well. Web tools might be accepted to a varying degree, but very few will be able to use HTML directly.

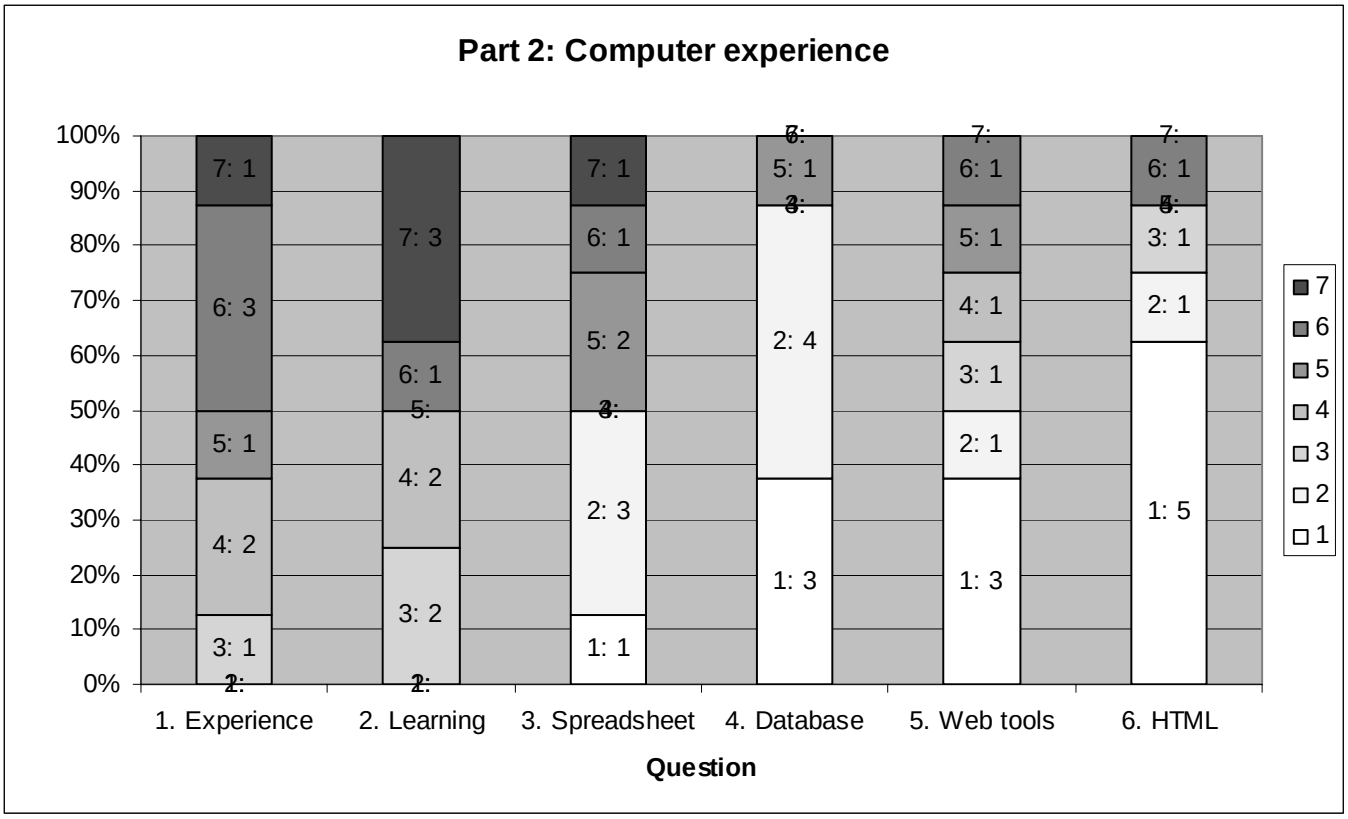

Figure 5.2: Computer experience

Merchant Interface 2.0

Seven clients answered part 3 of the questionnaire. There were noticeably many marks. This may be because the client did not have to make any trade-offs, but could check as many boxes as he or she wanted to. This could maybe have been done in a better way, like rating features on a scale, or setting a maximum limit to the number of marks. Still, our intention was to get a basis for filtering out less relevant features, meaning we can focus on the extreme results for prioritizing items.

All clients wanted to see an order history and statistics on what visitors search for in their shop.

On the other hand, there was little interest in an on-line manual and support for several languages and currencies.

The rest of the items were relatively equally scored and it was left for us to determine what should and should not be placed in the requirements specification.

Some of the lower prioritized items with a low cost may be implemented anyway, and opposite, higher prioritized items with a high cost may be left for a later release.

5.2 Requirements specification

The requirements gathering described in the previous section resulted in a requirements specification. It is summarized below. The entire specification is available in appendix B.

5.2.1 Sources of requirements

So what were the sources of the requirements? Many requirements came straight from the previous system, and although the back-end had no user interface, the tasks involved spoke for themselves.

We remember from section 5.1 that the User Questionnaire consisted of three parts:

- Part 1, *Environment*, generated a few requirements concerning performance and usability.
- Part 2, *Computer Experience*, introduced the requirement of a WYSIWYG-editor, as opposed to a plain HTML-editor.
- Part 3, *Merchant Interface 2.0*, indicated which features where most desired, and thus gave us directions towards selecting and prioritizing them.

Still, some of the requirements were not discovered until when the first user interface prototype was tested, as noted in section 6.3.

As stated in the introduction, “we need to be assured that the final system reaches a certain level of usability“. The Usability Engineering methodology is focused on doing just that, and was therefore adopted for this project. This is the origin of the usability related requirements, which are also included in the specification.

User profiling is the first activity in Usability Engineering (see chapter 3). Parts 1 and 2 of the User Questionnaire were the basis for the user profile, or “Users of the system”, which is summarized below as a part of the requirements specification.

5.2.2 Summary

This section summarizes the requirements specification. The requirements are divided into two main categories; functional and non-functional. The requirements are further divided into category of operation, and are summarized below.

Users of the system

Two kinds of users will be using the system; the clients and the super administrators (SAs). Clients are reasonably experienced with computers, and probably enjoy learning new systems. The super administrator is involved in all parts of the system, and has good knowledge of both the system and the task domain.

Functional requirements

As mentioned in chapter 3, the functional requirements describe interactions between the system and its environment, or the systems reactions to certain stimuli.

Product and price related requirements

The client must be able to add products to the shop. Products exist in SD’s database, and the client needs convenient methods of finding or browsing products, that can then be added to the shop. Non-relevant products can be hidden once and will not be listed again unless requested.

The client should be able to create own categories, that products can be sorted into. In addition, relevant SD categories can be selected, and linked to own categories. This has a threefold advantage. First, only *relevant products* are listed. Second, when products are added, they are *automatically categorized*. Third, this enables a client to *subscribe* to newly registered *relevant* products.

Another set of requirements specify actions that can be performed on products already in the shop. This includes setting attributes and categories, selecting related products, and uploading and selecting up to four product images.

Customer related requirements

The client should be able to create customer accounts. Customers' accounts can be organized into categories. A customer category determines the price among other things.

The client should also be able to send newsletters to customers who have agreed to receive them. The newsletter can be in plain text, or in HTML, in which case a WYSIWYG-editor should be used.

Price related requirements

These requirements state that campaigns can be defined with various conditions and types of results. Exceptions to the standard price can be defined.

Administration requirements

The SA must be able to start up an MI2 instance for a particular client. Then, the client should have a possibility to create several login accounts that can be used to enter the MI2.

The client should be able to edit page contents for different types of pages, including defining style sheets and layout. In the simplest version, pure HTML-code is edited, but preferably, a WYSIWYG-editor is implemented.

Statistics related requirements

The client can choose to view recent or common search terms typed by users to find the web site, as well as search terms used to find items in the web shop. The client should also be able to track the response of a newsletter by counting the number of clicks for each link. Lastly, the client should be able to view a list of recent orders.

General requirements

The shop front-end shall on demand support viewing in different languages selectable by the visitor. Thus, the MI2 needs to support translation of all its contents. This requirement is very complex, and has a low priority, so is to be implemented in a later release.

Non-functional requirements

Non-functional requirements focus on every aspect of the system *other* than the functional. They restrict our choices in constructing the system. Notably, some of the requirements below are not verifiable. They serve as a guide to the design, rather than provide quantitative measures to verify. In larger projects however, verifiability is vital to assure the customer that the requirement has been met.

Usability

To ensure proper usability, the method of developing the user interface should involve *iterative usability testing*. In addition, “Discount usability testing” should be performed to gain user feedback, as well as *expert reviews* using the “8 golden rules”.

The MI shall provide short on-line help texts for parts of the interface that are not self-explanatory for the clients. As a further help, a start-up guide shall be accessible on-line for the client, which helps the client to perform common tasks in a recommended order.

The interface should be usable with both Internet Explorer 6 and Mozilla Firefox 1.5 with a minimum screen resolution of 1024x768 pixels.

Performance

At least three users from the same client should be able to work on MI2 simultaneously, using their own login account. The response times of the interface should be about the same over the whole interface.

Compatibility with existing environment

These requirements state that the database engine for the system is Microsoft SQL Server 2000 or later versions, and that the system will be deployed on a Microsoft ASP.NET 2.0 platform, using IIS6.

6 Interface design

This chapter summarizes the design process, and shows some examples of how the design evolved. The final design is shown in chapter 7.

6.1 Method

The basis for the design is the requirements specification from the previous chapter. We remember that the interface to the clients is web based, so with the exception of a few non-functional requirements, all the requirements should be satisfied by a set of pages. Every page design has a purpose, and together they need to fulfil all the relevant requirements. It is stated together with each page design what requirements are satisfied with it, which lets us trace each feature back to the requirement that mandated it.

Two requirements in the specification specify how the design should be developed. Non-functional requirement NR001 states:

> To ensure proper usability, the method of developing the user interface should involve iterative usability testing. The phases of designing and testing the user interface should be iterated at least three times, and finish when fix point is reached, where the improvement in subjective satisfaction of the users are less than 10% between the iterations.

Requirement NR002 says:

> "Discount usability testing" should be performed to gain user feedback, as well as expert reviews using the "8 golden rules".

We can see that these requirements will be fulfilled if we follow the recommended activities for *Usability Engineering*, as described in chapter 3. This includes iterating the phases design, testing and evaluation.

As we learned in chapter 3, the Software Engineering practice *throw-away prototyping* has the purpose of exploring feasibility, both during the requirements elicitation phase and the design phase. In Usability Engineering however, prototyping is used in an iterative development process to evaluate usability levels.

In my approach, I will to combine these two ideas, and thereby attempt to achieve the desired levels of usability, as well as evaluate if the set of requirements are complete.

6.2 Overview

It is clear from the requirements specification what the system needs to do. I will attempt to give an overview of the pages that are thought of for the first design.

Table 6.1: Overview – Pages

Class	**Description**
SDSearch	This page should provide an interface for searching for products by entering search terms and optionally specifying search constraints.

Class	Description
SDProducts	This page will display the results from searches made with SDSearch, and will additionally be used for browsing the product catalogue. The client should be able to add, remove or hide any of the displayed products.
Products	The Products page should list products that have been added to the shop, and should provide links to editing or removing a product, and to setting associated products.
ProductDetails	This page should contain all that is needed to edit product details.
Upload	The upload page lets the user upload, browse, preview, and choose images.
Associations	The Associations page should let the user choose related products to some specific product.
Categories	This page is for creating and maintaining categories.
Pages	This page gathers all the web pages that the client should be able to edit. The user can edit all pages, and also add and remove custom pages.
Page	This page lets the user edit the contents of a page, preferably in a WYSIWYG-editor.
Accounts	This page lets the user create additional user accounts for MI2.

6.3 Iteration 1

I decided to start the design process by sketching a paper mock-up of a suggested user interface. This mock-up could then be used for usability testing and evaluation in preparation of a redesign.

6.3.1 Design

The basis for the design was not only the requirements specification, but also knowledge of the domain acquired during the requirements gathering. At this point, a task analysis would typically be useful. The goal was to create a design that would stick to common patterns and a workflow for the users that would be as simple and self-explanatory as possible, yet powerful enough to support all the required features. We must recognize that at some point, all potential users will be first-time users, and the system must be designed so that they easily can get started. At the same time, any user that decides to continue to use it could get bored if there are no shortcuts, or expert features. This must be taken into consideration during the design.

The process of developing a design is creative, and is about finding a good solution to a known problem. Thus, each requirement can be viewed as a problem, and the design as a solution. The design does not immediately have to fulfil all requirements, as long as the

final design does. A few examples are shown below of the first real prototypes along with a description of each. The rest of the prototypes were developed in a similar manner.

SDProducts

If we again look at the summary of the first few requirements that states:

> The client must be able to add products to the shop. Products exist in SD's database, and the client needs convenient methods of finding or browsing products, that can then be added to the shop. Non-relevant products can be hidden once and will not be listed again unless requested.

The solution I had in mind was to have a separate search page (SDSearch), where a result list would be displayed on a separate page (SDProducts), see *Figure 6.1*. The result page could additionally be used to browse products, which is a special case of searching without text strings. Each item is repeated to form a list of products. Now we can find products.

To add products to the shop, we could have either buttons or a method of selecting them. I chose to have checkboxes so that the action easily could be undone to prevent mistakes. A product that is already in the shop is clearly marked by a text, and has a checkbox for removing it instead of adding it. Similarly, a checkbox for hiding and unhiding a product is also available. The result list can be split over several pages, and filtered by conditions set at the top of the list. However, grouping or filtering by product categories is not yet supported in this design, so it will need to be refined.

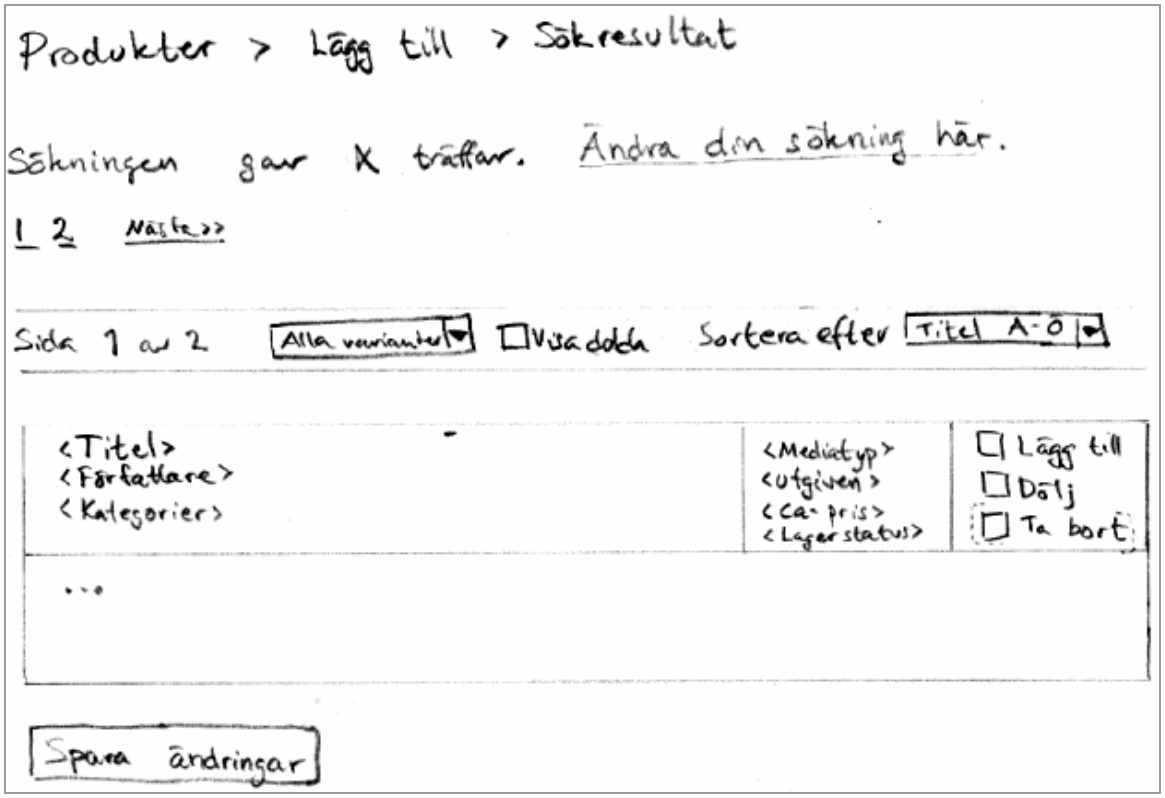

Figure 6.1: Draft – SDProducts (viewing search results)

Products

Two other requirements state actions to be performed on existing products, namely a way of finding them, and editing them. At first, I thought the design above could be reused, but

then I quickly realized that this list contained a lot more information than what was necessary for just finding a product. This spawned the idea of a slim list, organized in a tree structure by category, with a quick search feature. The design is called Products and is shown in *Figure 6.2*. It is possible to choose to display either hidden, all or just imported products by the drop down box at the top. There are buttons on each item for editing, removing and setting associated products. This page satisfies requirement *FR004: Finding existing products*. In this version, categories are represented as nodes in a tree.

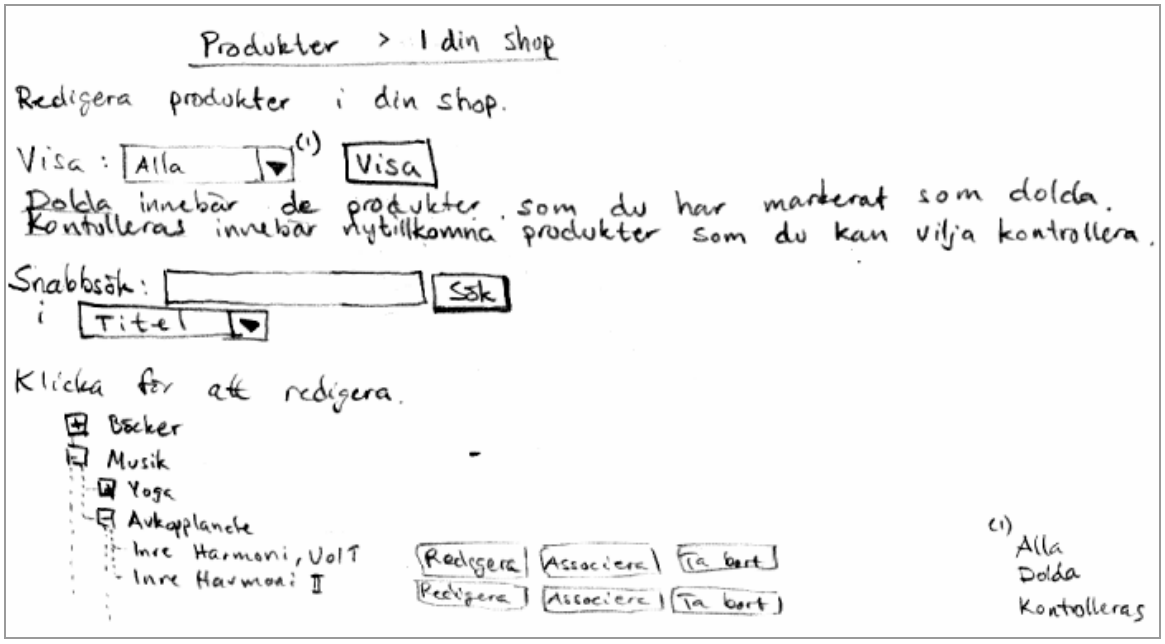

Figure 6.2: Draft – Products

Quick redesign

As the design above was completed, I did a quick walkthrough with one of our clients and asked for some input. I got some very valuable suggestions, and further insight into the problem domain.

The major change was the way that “own” categories were assigned when products were added to the shop. Each product is supposed to belong to one or more categories to simplify navigation. Previously, categories would somehow manually be selected and added for each product after it was added. This extra step required made the procedure seem tedious and prone for mistakes, such as forgetting to assign categories. The new design would do this automatically. The idea was to have “links” between SD categories and “own” categories. There were two advantages. First, when a product is added, the links would be used to find which “own” categories correspond to the existing SD categories of the product. This way, links are established once, and “own” categories are applied automatically. The second advantage is that the selected SD categories can be used to filter only products from selected categories when browsing and searching for products, making the list more relevant to the client. This resulted in a new requirement; *FR023: Choosing subscriber categories*, along with a new design, namely the SDCategories page (see below).

The second very important suggestion was to have a “checklist”, or start-up guide available. The checklist would not be as dense as a manual (which our clients seemed less

interested of), but instead focuses on explaining common tasks and suggesting what to do, and in which order, short and concise. This idea spawned the requirement *NR004: On-line start-up guide*, and the new Checklist page that would fulfil it.

Because of this feedback, I decided to postpone the usability test and do a quick redesign first, and came up with the following new designs.

SDCategories

The newly developed design is shown in *Figure 6.3*. It provides the link between SDs' categories and the clients' own categories. An SD category to the left is selected by a checkbox. If this box is checked, a corresponding category must be chosen to the right from a drop down box. This design fulfils *FR023: Choosing subscriber categories*.

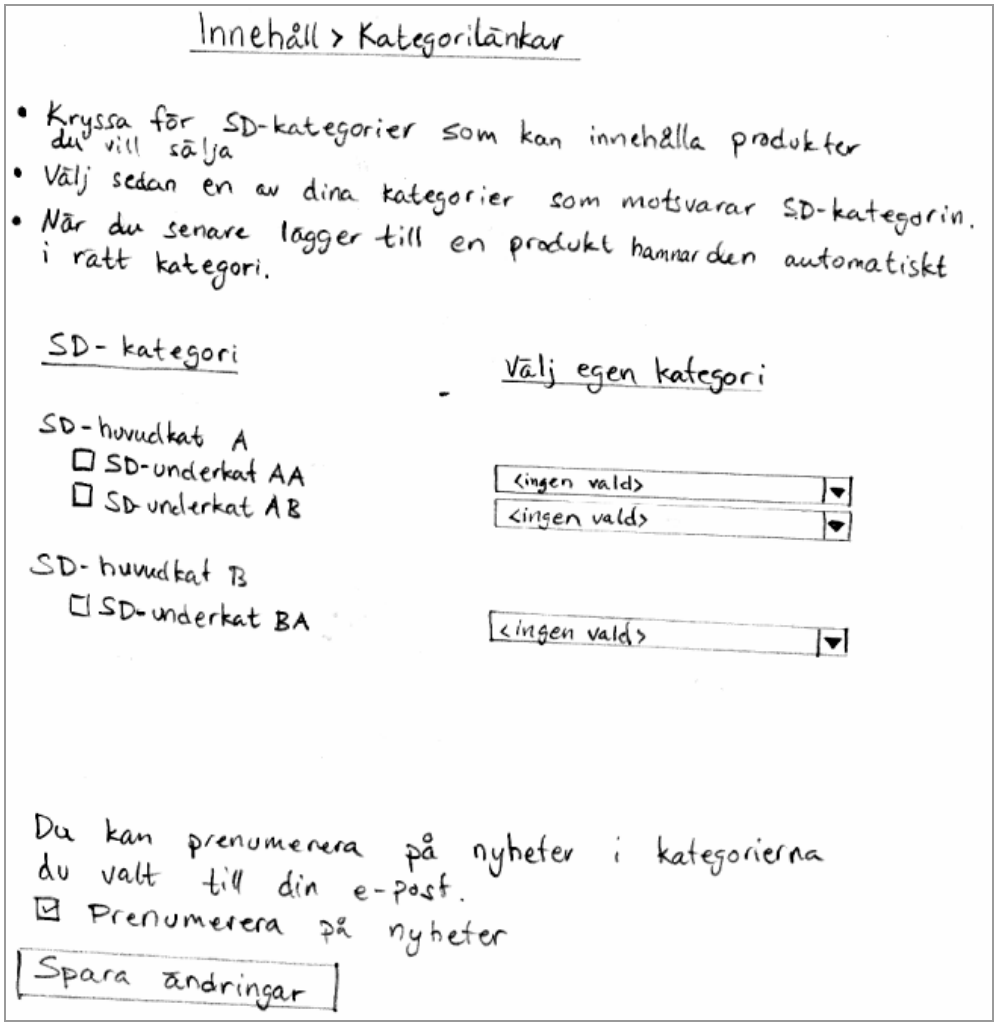

Figure 6.3: Draft – SDCategories

SDSearch

The new category links made it natural to add filter conditions to the previously mentioned search page. A checkbox "In selected categories only" replaced the previous drop down box that only supported searching in one category.

SDProducts

Since the first design of SDProducts did not completely meet the requirements, the redesign made sure that grouping and filtering was finally supported, as specified in

FR001: Finding new products. The filtering mechanism was comprised of checkboxes and drop-down boxes that would allow the user to change the filter on the fly, with options such as showing products “In selected categories only”.

6.3.2 Usability testing

Usability testing was done as *discount usability testing* (see chapter 3), using the paper-prototype just developed. The user was asked to do a set of tasks described on a sheet of paper. The tasks were created to cover the most common activities from a start-up perspective, by setting up parts of the shop from scratch. A typical task looks like the following:

> You would like the price of the products to be 10 SEK lower than the recommended price.
> Please add all the books from your publishing house to your shop.

This particular task includes finding the search page and which search field is best to use. Following that is browsing results, and adding products with a specified price.

The user was encouraged to use the start-up guide (checklist) to get further information, since this would be a typical usage in the completed system. His behaviour was observed and noted as well as his comments. The test supervisor was responsible for shuffling papers around as the client interacted with the system. He also had to fill in dynamic contents such as error and confirmation messages as well as display or explain the results of the client’s actions.

Usability test 1

The test user was a 45-year-old male person that had been using the previous version of the software, and thus had good domain knowledge.

At the end of the test, most of the prototypes were cluttered with comments on post-it notes.

Task 1.1 Choose own categories (Categories)

The task of adding categories posed no problem to the client. He said that he was looking for examples of what the categories could be, but found none. He was also looking for a familiar way to return to the checklist, such as a “Return to checklist” button. Instead, he used the back-button on the browser. He also wondered why there was a sort button for the sub categories but not for the main categories.

Task 1.2 Choose categories at SD (SDCategories)

With the task of selecting categories at SD, the client got a bit confused because the captions were so similar. He suggested calling the former “Own” or “Your categories” in the interface as opposed to “SD categories”.

The task description at the top of the page was not easy to scan through which delayed the task a bit. This partly made it more difficult to understand the option “<none selected>” in the drop down box.

After fully understanding what the whole procedure was for, he commented that it was counterintuitive to have SD categories before own categories. I noted the comment, but later explained that making it the other way around would severely complicate the design because of the 1:N relation that would then be reversed in the interface.

Task 1.3 Add own products (SDSearch, SDProducts)

The greatest problem appeared at task 1.3.1 where the client should add all products from his own publisher. He clicked the link for browsing categories. Despite several attempts of using the filtering mechanism for locating the products, he was just unsuccessful for a long time and got frustrated. Eventually I gave the advice to use the search page instead, and from there on there were no major problems.

When browsing for news, he thought the term “Age” was unintuitive, and that “Published” might be used instead. He also thought there were too many options in the Age drop down box.

Task 1.4 Campaign (Campaign)

Here there was some confusion because “the second stage of adding a campaign looked similar to the first, so what was the purpose of the second page?” The client also sought an indication of where in the process he was currently, like “Stage X of Y”.

At one point, he wished to select a product, but was confused by the textbox next to the “Choose…” button. “What is the textbox for?” “When do I click choose?”

When entering a second campaign, after picking a date, he could not grasp what the checkbox “The campaign is active” was thought to do. “If I choose dates, then I don’t want it to be active at dates other than specified. If it is for removing a campaign, then I expect to find a remove button in the list I get at the end. “

When selecting a new price for the product, he wished to see the default price for comparison.

Task 2.1 Extra product information

The test had so far taken over an hour, so it was decided to end it there, and do the questionnaire instead. We had gotten a large amount of feedback on the most important features anyway, so the test was a success.

User comments

The client commented that he felt frustrated at one point, and that going back was not very intuitive. When a page posts back to itself several times during some operation, the back button appears to have no effect. This is due to the use of server-side script that ASP.NET executes for many operations.

The use of the second step of the campaign wizard was not clear to him.

When a button said “Choose…” it sometimes refrained him from clicking on it, because he did not always know what it would do.

6.3.3 Evaluation

After having performed just one test, we had gotten so much feedback that it was suitable to do a redesign without further testing.

Questionnaire

After the test, the *User Satisfaction Questionnaire* was handed out, see *Appendix A – Questionnaires*. The purpose of the survey was to get a measure of the users' satisfaction of the system. The result gives a hint of the *subjective* usability level of the system, and is useful in comparing results from different stages in the development. An example question is shown below:

```
5.6 Error messages            unhelpful              helpful
                                 1  2  3  4  5  6  7  8  9      NA
```

The distribution of answers is shown in *Figure 6.4*. Each bar corresponds to a number on the scale between one and nine, and represents the total number of answers for that number.

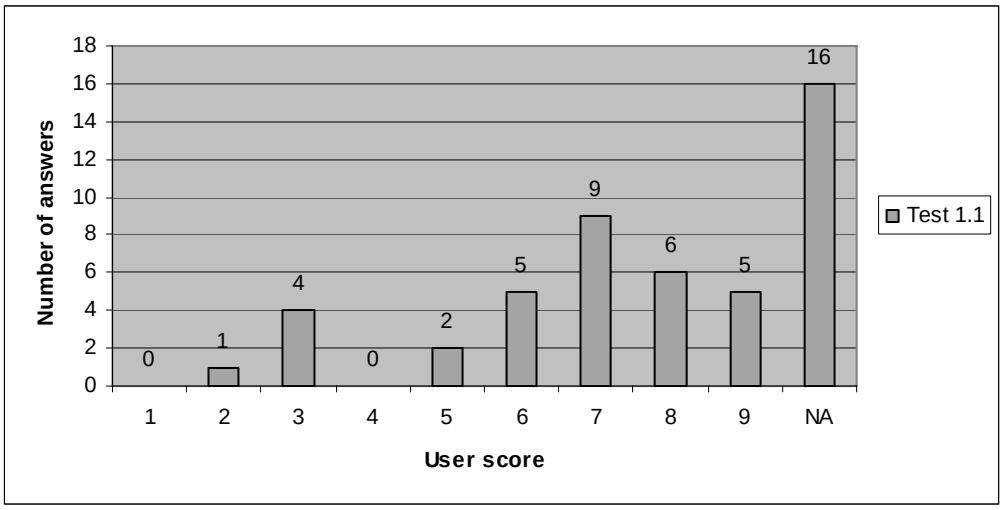

Figure 6.4: Answer distribution – User Satisfaction Questionnaire 1

The average response was 6.6 points, on a scale between one and nine, at the end of the first iteration with a paper prototype.

Notably, there are many unanswered questions. This is due to that many questions are meant for the implemented system, and are not applicable to the paper prototype. This difference must also be taken into account when comparing results with those of the implemented system later on.

Most scores circled around seven as we can se in the figure. It is therefore interesting to investigate the cause of the lower scores. Some of the deviating low results are listed here, along with a plausible explanation:

- 3.2 "Overall reactions to the system" (frustrating – satisfying): 2 points.
    - By mistake from the test supervisor, there was a hard-to-understand message at some stage when using the "Back" button.
- 4.4.1 "Next screen in sequence" (unpredictable – predictable): 5 points.

    - Due to the confusing order of screens when creating a campaign.
- 4.4.2 “Going back to the previous screen” (impossible – easy): 3 points.
    - Same as 3.2.
- 5.4 “Messages which appear on screen” (confusing – clear): 3 points.
    - Same as 3.2.
- 6.2 “Exploration of features by trial and error” (discouraging – encouraging): 3 points.
    - Same as 3.2. Possibly also due to frustration at task 1.3 when looking for products to add.
- 6.4.2 “Steps to complete a task follow a logical sequence (never – always): 3 points.
    - When creating a campaign, the second step seemed to be same as the first.

Suggested redesign

After the evaluation, a redesign was suggested with items such as these:

- Add “Sort alphabetically” button for main categories in “Categories”
- Make information on pages bulleted and scannable.
- Add a stage indicator to the “Campaigns and prices” page, such as “Step X of Y”.
- Clarify the task formulation of the checklist.

Continuation

The results of this test were used in a redesign, which marks the end of iteration 1.

6.4 Iteration 2

This design was again made as a paper mock-up.

6.4.1 Design

Some of the proposed changes from the previous evaluation were applied to the design in this iteration. Here are some examples of actual changes made:

Categories

- The page title was changed from “Categories” to “Your categories”.
- A sort button for main categories was added.
- Buttons that perform actions on rows were moved to the actual rows to be consistent with the rest of the design.

SDProducts

- A link was added pointing to the search page.

General

- On-line help at the top of each page was reorganized to a bulleted format to make it more scannable.

6.4.2 Usability testing

After the first redesign, the new paper prototype was tested by two employees within the company. One had little computer experience and the other had extensive experience.

Usability test 2.1

The user for this test was a 44-year-old woman. She had little computer experience, which showed in that she was a bit insecure in many tasks. Still she managed to do all the tasks without any major problems.

Task 1.1 Choose own categories (Categories)

Adding categories was no problem, even though she did look at one of SDs' other web pages for examples of categories.

Task 1.2 Choose categories at SD (SDCategories)

After checking a couple of SD categories, she was expecting to see a list of products instead of having to pick an own category. Maybe the short instruction was misleading.

Task 1.3 Add own products (SDSearch, SDProducts)

Just as the previous participant, she went immediately to the browsing page to look for products by a particular publisher. It took a minute to realize that maybe she should be using the search page instead.

After finding a list of 20 products, she wanted to add the first one. She changed the price but was not sure what to do after checking the "Add" checkbox. Eventually she found the save-button at the bottom, and clicked it. The feedback on the page made her confident to add the rest of the 19 products. The meaning of the "Ignore" option was not clear to her, on which she commented.

When doing a search for a recent product, she lacked the option for selecting publishing date. She made the search anyway, and found the date option on the result page instead.

Task 1.4 Campaign (Campaign)

At the second page when adding a campaign, she wanted to know what she had selected on the previous page, so she used the previous button to get a glimpse and then went forward again. Thus, it might have been useful to state every previous choice for easy overview.

Task 2.1 Extra product information (Products, ProductDetails)

When looking at "Products in your shop" she was a bit confused about the options in the "Type" drop down box. Improved on-line help would probably have been useful.

User comments

She commented that she would have been more at ease with using a real computer screen instead of the paper mock-ups.

Usability test 2.2

The user for this test was a 33-year-old man and had extensive computer experience. The tasks were quickly performed and many options appeared self-explanatory.

Task 1.1 Choose own categories (Categories)

The only issue here was that the user did not see the sort button until afterwards, and thus did the sorting manually. This would probably have been clearer on a real screen.

Task 1.3 Add own products (SDSearch, SDProducts)

For adding all his "own" products, he first looked at the browsing page, but found no options to find them there. He then quickly went to the search page, and did a successful search. Because the search was a 100% match, he was looking for an option to add all products, but there was no such option. He did not change the price on any product at this stage. Instead, he had to edit each product manually afterwards and set the price 10 units lower, as the task description stated.

Task 2.1 Extra product information (Products, ProductDetails)

When adding an extra product image he uploaded the image correctly and it was displayed for preview. Then he mistakenly clicked "Close window" instead of "Choose", and the image was not transferred to the product. So he repeated the procedure and made it right the second time. The placement of the choose button was apparently an issue here.

Task 2.2 Category text (Pages, Page)

In this task, the user first looked at the page "Own categories" and tried "Edit", but was unsuccessful, since this only edited the name of the category. He went back to the checklist (which he had overlooked before), found the correct page, and finally managed to add text to the category page.

User comments

The user commented that in many pages there was a save button. It is possible, and even likely in the beginning, that the user might miss these buttons. He requested some solution to this problem.

I showed him the other designs as well, and got the feedback that the newsletter could easily be adapted so that it would be possible to type HTML code as well as text. Advanced users could design the letter in another editor and paste the code into the system.

In addition, it is necessary to be able to send a test copy of the newsletter before doing the actual broadcast.

6.4.3 Evaluation

After these two tests, we had gathered sufficient material for the next redesign.

Questionnaire

The User Satisfaction Questionnaire gave high results this time. The users responded on average 8.4 and 7.4 points, with a combined average of 7.9 points (between 1 and 9). This is substantially higher than the first test, so we can conclude that the redesign was a major improvement.

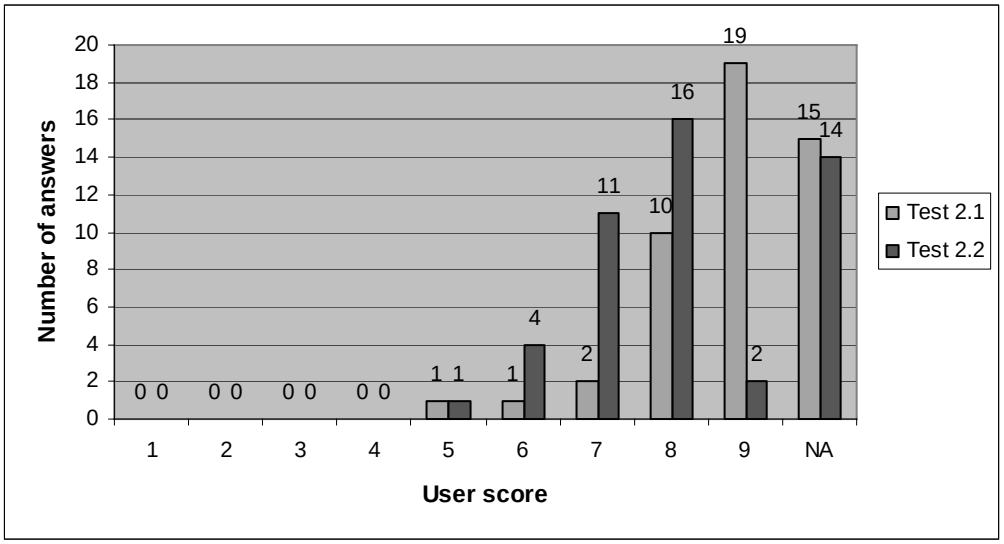

Figure 6.5: Answer distribution – User Satisfaction Questionnaire 2

Since there are two users this time, we look at each individual questionnaire and find for example that Test 2.1 gave 4 scores between 5 and 7 that were much lower than the others. Again, we try to determine the cause of the deviating lower scores:

- 3.4 “Overall reactions to the system” (difficult – easy): 6 points.
    - The user was stuck at some point in the tasks. Knowing that there is a solution to the problem at hand, but not finding it might make the user think the system is a bit difficult to use.
- 4.4 “Sequence of screens” (confusing – clear): 7 points.
    - When creating a new campaign, the options made so far were not displayed, and the user had to step back and fourth, which might appear a bit confusing.
- 6.1 “Learning to operate the system” (difficult – easy): 7 points.
    - Not all tasks went as expected, and thus the user might feel that learning the new system is not always easy.
- 6.4 “Remembering names and use of commands” (difficult – easy): 5 points.
    - Selecting a product for a campaign meant that the user had to remember the product id and then manually enter it.
- 6.2 “Exploration of features” (risky – safe): 5 points.
    - I am unsure of the cause for this.
- 3.1 “Overall reactions to the system” (difficult – easy) & (rigid - flexible): 6 points.
    - Probably due to a combination of the relevant issues above.

On the positive side, part 6.1.1 “Getting started” (difficult – easy) and 8.1 “Information from the checklist” (confusing – clear) got nine points from both users. This is likely thanks to a well-needed and designed checklist.

Suggested redesign

Many fixes were suggested to remedy the problems discovered during the evaluation. In particular, the meaning of category links (SDCategories) still seemed to be unclear. Here are some examples of the suggestions made.

- SDCategories: Clarify the purpose of this task in the short introduction.
- Campaign: Show the options selected so far to reduce memory load.
- SDProducts: Add an option to add all products on the page, which in reality would mark all products as “Add”.
- Upload: Move the “Choose” button next to the “Close window” button, and call them “Choose image” and “Cancel” respectively.
- If it is feasible, construct client script to prevent a user from leaving a page without an option to save it first.

Continuation

Since the test users experienced some problems in iteration 2 that might occur due to testing a paper mock-up and not a system on screen, and the fact that no major issues were discovered during the last iteration, it was decided to continue usability testing after implementing the redesigned system in software.

6.5 Iteration 3

At this stage, the project was implemented and deployed for testing. All features that the user would come in contact with were implemented but not yet fully tested. Therefore, some software faults were discovered along with some usability problems.

At this point, we abandoned the *throw-away prototype*, and began developing an *evolutionary prototype* (see chapter 3). This new prototype would then evolve into the final design shown in chapter 7.

6.5.1 Design

The basis for the computer design was the paper mock-up from the previous iterations along with the suggested redesign. Of course, there is no one-to-one mapping from a freely designed paper mock-up to an implementation built on an existing platform as ASP.NET with its built-in limitations. Therefore, some features were straightforward to implement with standard components, while others were not feasible. Some implementation issues are discussed in *8.3 Implementation issues*, such as problems with code size mentioned below.

Here are some interesting examples of the design changes for this iteration:

Upload

- The original design intended a TreeView to be used for selecting catalogue. As this would have produced a vast amount of HTML-code, it was replaced by a drop-down box.
- Related buttons “Choose” and “Close window” were grouped together and their caption changed to “Choose image” and “Cancel” respectively to better reflect their purpose.

SDCategories

- Initially, all SD categories were listed, but it turned out to be too many (close to 160) which made the page to large (>100kB). Therefore, smaller groups of sub categories are now shown only on demand.

Associations

- The page was redesigned so that a new window would not have to be opened to select related products. Instead, clicking a title would select a starting product, and from there one would check related products as before, but now in the same window.

Products

- When the product list was to be implemented, it showed that the original idea with a ASP.NET TreeView was both slow to use and was not flexible since buttons was not supported. Instead, a Repeater was chosen for its flexibility.
- Grouping by category was done with separators titled with the category name.
- Paging was added manually, since the built-in component did not support this directly.

Page

- A WYSIWYG-editor was implemented with a third part component.

6.5.2 Usability testing

Two new tests were set up with a couple of employees at the company. The setup was a bit different from usual however. Since I could not be on the physical location, I set up a remote session using the “Remote assistance” feature in windows combined with an open phone line. “Remote assistance” allows a user to connect to another computer and observe the desktop. A disadvantage of this type of setup is that you cannot see the users’ physical expressions and gestures, but have to rely on a dialogue and watching the screen only. It is still noticeable when the user is stuck since there is a long silent pause in the task, which as usual leads me to ask what he or she is thinking now. It did however work out fine, and additional valuable feedback was acquired.

Usability test 3.1

This user had some computer experience and had the belief that nothing could go seriously wrong by using the computer, so she claimed to be unafraid of exploring new features.

Task 1.1 Choose own categories

Where the user shall add main and sub categories there are two text boxes and buttons respectively. The user filled in the main category box and immediately did the same with the other text box. She clicked “Add” for the sub category only, which caused unexpected behaviour of the program. When I asked why she did so, she replied that she had seen the first “Add” button, but thought that both would be added at the same time, as she was used to from somewhere else. This usage case should be considered as an occurring variant.

When she was editing a row in the table, she automatically pressed the “Enter” key to save the row. However, this did not save the row. Instead, a red star appeared next to an unrelated text box, and she gathered that this meant something was wrong, and eventually found the small save button thanks to the legend at the bottom. An important principle of usability says that we should stick to common patterns and designs. This is one such example.

Task 1.2 Choose categories at SD

This task went just as expected, but she seemed to have misunderstood the meaning of these category links. Maybe the name was misleading because she thought it would mean that customers from SDs’ web shop would somehow find their way to her shop as the name “links” implies. It took her a few minutes to figure out that it had something to do with browsing the product catalogue, even though the meaning was mentioned in the short information at the top of the page. There was a direct link to a page that showed the result of the filter. She did try it a couple of times, but could not make sense of what it meant.

Task 1.3 Add own products

Unfortunately, the search function failed to give any results for this test. Nevertheless, it was discovered that the caption of the search text fields was not very clear, since the user used the wrong field when searching for titles by a specific publisher.

The user did not understand how to find news in the categories she was interested in, even though she had actually been to the correct page before. In the end, she found the news, but had to filter the categories manually.

Task 2.1 Extra product information

The task of adding an image to a product went ok until after uploading it, where she closed the window instead of selecting the new image. This was probably because the select button was hidden by the confirmation message that pushed it down, and the pure size of the very large sample image. It also became apparent that the window itself could be resized in Mozilla Firefox, but not IE.

After some help, she managed to select the image, but then went on to the next task in the test without saving changes to the product. This recurring usability problem needing attention.

She also tried the Associations, but it probably did not mean what she expected, so there was some confusion around the meaning, but not the function of this page.

Usability test 3.2

The user for this test had extensive computer experience. The tasks were quickly performed and many options appeared self-explanatory.

Task 1.1 Choose own categories

The user was about to edit a table row, and was probing the functions of the small icons by pointing the mouse and reading the tool tip, but it appeared to be missing for the edit button. Nevertheless, this was not a problem, because he guessed it would be the correct button and tried it anyway with success.

Task 1.3 Add own products

While adding products, the user commented that it would be useful to see the purchase price as well as the recommended price. In addition, one might want to read a description text or see a larger image than the small thumbnail for a product.

He discovered that the page did not remember the number of results per page, which he repeatedly had to change to his desired 100 per page.

Task 1.4 Campaign

When adding a certain type of campaign, he wanted to type a description for it as for the other ones, but this option was not available due to a deliberate restriction that might actually have been useful in this case.

Task 2.1 Extra product information

A problem was discovered when saving the page. He had just edited a row in the GridView of Attributes, but not saved it. When saving the whole page, the row was not saved as he guessed it would have. He corrected it, but it pointed out an issue that could occur on other places, when a GridView is part of a larger edit form.

Task 2.2 Page contents

When the user was creating a new page, he wrote the name of the page but left out the file extension. This gave an error message after which he corrected it and commented “Couldn’t it figure that out by itself?” It probably could, since there is only one available extension, which could be the default.

After adding that page, he got confused for a moment since the new page appeared *above* the input box, which breaks a common pattern.

User comments

He tried the newsletter feature with success. After saving the newsletter, he wondered how he could preview it or get a sample copy sent to him. There was no such option, but obviously, this would be a very useful feature to have.

6.5.3 Evaluation

Questionnaire

After the test, the User Satisfaction Questionnaire was handed out as usual. See the distribution of answers in *Figure 6.6*.

The average response was 7.4 and 6.7 points respectively, on a scale between one and nine. This yields a combined average response of 7.1 points at the end of iteration 3.

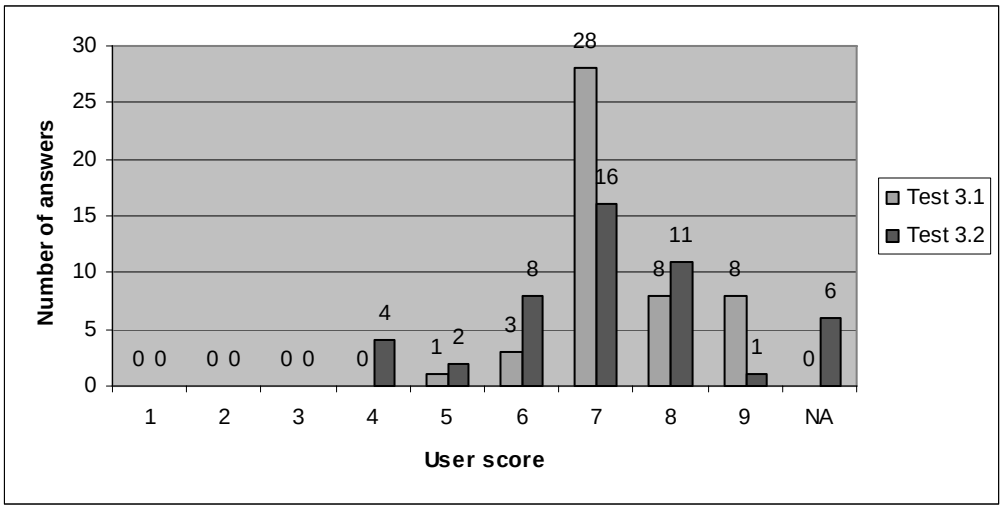

Figure 6.6: Answer distribution – User Satisfaction Questionnaire 3

One item that got a low score was "4.4.2 Going back to the previous screen", with 4 points. One user noticed that clicking the "Back" button on the web browser only stepped back through *steps* such as adding and editing records on the *same page*, instead of backing directly to the *previous page*, as he intended. This is a serious usability problem, as we are used to the way the back button works, but suddenly are forced to find another way to navigate. I would preferably see a solution to this issue with ASP.NET[6].

Another item with a low score was 6.1.2 "Learning advanced features" (diffucult – easy) with 4 points. No comment was given.

On the positive side, "8.1 Information from the checklist" got 8 plus 9 points (clear), which is very good, since this is a very important feature for the usability of the system.

User comments

Some of the users' comments are presented here.

- "An easy system, although I usually don't spend too much time browsing manuals. And 'Learning by doing' seems to work, even here."
- "I sometimes was stuck between the simple task and the need just because the name of a file (link) was different than what I expected, but in all, it's all about what you are used to."

[6] Later on, Microsoft announced the "Atlas" framework add-on, which potentially could solve the navigation problem, since it utilizes client script to refresh only parts of a page.

- "… I've learned that the best thing to do is just to go back and do it again, being more concentrated on the task."
- "I learn easily and I find a lot just by looking. Unfortunately, I often skip manuals, unless it's a delicate device that clearly warns for such behaviour. Since it's impossible to ruin something on a computer, I've found that I learn more by thinking, reacting and doing."

The user apparently learns mainly by "trial and error", and takes less notice of help texts.

Suggested redesign

Here is a selection of some important suggestions for the redesign:

- Link together add buttons on the Category page so that a main and sub category can be added at the same time if both text fields are filled in.
- Clarify the meaning of category links (SDCategories), again.
- Can there be a safety net so that when the product details page is unloaded, but some contents have not been saved, there is a confirmation prompt first?
- Make sure there is a tool tip on every icon button.
- Product result list: Make it possible on demand to read full description text and see full size image.
- Make "number of results per page" persistent over a session.
- When a page is being saved, but at the same time a row in a GridView is in edit mode, the GridView should first perform a save instead of dropping the edited contents, since this is what the user intends.

Continuation

So far, with discount usability testing, much feedback has been gained. In the interest of finishing soon and testing from a different perspective, it was decided to continue the usability testing by expert reviews.

6.6 Iteration 4

The suggestions from the previous evaluation were considered and incorporated into the design.

6.6.1 Design

These are some of the changes made to the design:

Categories

- It is possible to add a main category and a subcategory simultaneously by filling in both text boxes before submitting.

SDProducts

- "Number of results per page" is now persistent over a session.

- A popup window shows when the mouse points at an image, displaying a larger image and a full description text.

ProductDetails

- When the user clicks “Save” while a row in a GridView is in edit mode, the row is first saved, instead of being discarded as before.

Newsletter

- It is now possible to send a preview copy to the clients email address before sending to all recipients.

6.6.2 Usability testing

This time, usability testing is done as expert reviews.

Expert evaluation 1

The first evaluation was done by another employee at SD, who works as a computer engineer. No specific guidelines were followed. These are some of her most important suggestions:

SDProducts

- If you choose to ignore a product, it disappears from the list, but if you change your mind how do you reveal it again? On-line help would be useful.
- When browsing “In selected categories only”, but no category links are defined, there are no products and *no explanation*. Explain why and how to fix it.

Campaigns

- After step 1, we jump to step 3 without passing step 2, unless advanced options are selected. This non-linearity might be confusing to some users. The page numbering could be removed, and just keep the page headings.

Page

- The WYSIWYG page editor is somewhat advanced. Some of the users might be overwhelmed. A manual or walkthrough is probably needed.
- Strive for consistency: There is often a “Save” and a “Cancel” button at the top of the page in general, but not on this page. Add them for consistency.

General

- The feedback messages are just barely visible in bold text. Make them strike out more, maybe in colour.

Expert evaluation 2

This review was done in the form of a heuristic evaluation, guided by “The 8 Golden Rules” included in Appendix A. The method is described in section 3.2.5.

This review yielded a great number of suggestions. A few samples of different types of problems are shown below for illustration.

SDCategories

- Simple error handling: If a client category is chosen in the drop down box, but the user forgets to click the checkbox next to it, then nothing is saved, but there is *no message* about the mistake. If the checkbox was not checked before save, then it could be automatically checked and save the user from an unnecessary mistake.

SDProducts

- When grouping and sorting by "category, popularity", there are sometimes very many results in each category. Can excessive rows be reduced somehow? One would like a maximum limit on the number of rows per category.
- Offer informative feedback: When ignoring/unignoring products, there are no messages about what is happening.
- Strive for consistency: When sorting by "category, date added", the drop down box "published date" would better reflect date *added*, instead of date *published*.
- Enable frequent users to use shortcuts: If the user wants to read the description text, there is no immediate way to do so. Can there be a quick popup with a description and a larger image?

Campaign

- When pressing "Enter" in a text box, you would expect to go to next page, but instead you went to the previous page, since the "Previous" button occurs first on the form. If possible, make "Enter" move to the next page.

Associations

- Support internal focus of control: When a title appears in two different media types, they have the same name and the pair is thereby indistinguishable. A suggestion is to show the media type also.

General

- Strive for consistency: The edit icon sometimes opens a new page, and sometimes edits the row inline. There should be two different icons for these two actions.
- Strive for consistency: When doing inline table editing, I often pressed "Enter" to save the record, but this usually did something else, like submitting the first button on the page. "Enter" should confirm the action that you are doing as usual, to keep the confidence of the user throughout the interface.

6.6.3 Evaluation

When doing expert reviews, other types of issues are discovered compared to when doing discount usability testing. A lot of focus has been on making the interface consistent,

which is something that is difficult for a user to discover, since their focus is on solving a task, not on examining details.

Suggested redesign

Practically, all the issues found during evaluation were suggested for the redesign. Here are some samples:

Page

- Create a walkthrough for the WYSIWYG-editor.

SDCategories

- If a client category is chosen in the drop down box, but the checkbox was not checked before save, then automatically check it.

SDProducts

- Explain why there are no products when browsing “In selected categories only”, without any links defined.
- Add a popup for each product with an enlarged image and full description text.

Campaigns

- Remove step indicator, but keep page headings to keep the user informed of the progress.

Campaign

- Make the “Next” button default on the form, instead of the “Previous” button.

Associations

- Add media type after the title, to make duplicate titles distinguishable.

General

- Enhance feedback message appearance by making a coloured background and blinking a couple of times when the message appears.
- Create a separate edit icon for editing on a new page as opposed to doing inline editing.
- Pressing “Enter” when doing inline table editing should save the row.

Continuation

Theoretically, we could go on with usability testing for the interface until a fix point was reached as stated in the introduction. However, time was running out and the product was due to be delivered soon so this would be the last usability test before deployment. The final design is presented in chapter 7.

7 Final design & implementation

This chapter overlooks the final system design, starting with the main architecture, and drilling down through the most important modules, including the final user interface design. Following that are some examples from the formal design specification, including static, dynamic, and database structure.

7.1 Method

The design specification was not created at a single stage during the development, but instead evolved during the Usability Engineering process and prototyping detailed in chapter 6. The architecture was already decided by then, as shown below. At the end of iteration 2, when the prototypes were ready to be transformed into real user interfaces, the design specification began to develop. The static and dynamic structure was then defined, based on the prototypes of the user interface.

7.2 Main architecture

The overall structure of the system is illustrated in Figure 7.1

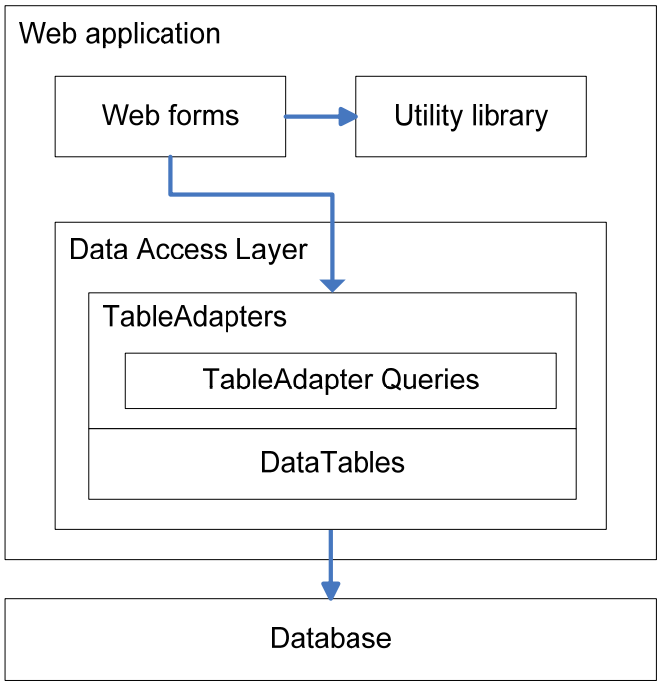

Figure 7.1: MI2 Architecture

The *web forms* make out the larger part of the system. There is one web form for each main functionality of the system. There are also very few dependencies among them. As described in chapter 4, application logic and user interface can be separated by using code-behind files. This leaves the user interface purely defined declaratively in .aspx-files, and additional code in a separate .cs-file.

The *utility library* provides common routines and enables code reuse. Routines for file access, authentication etc. are placed in the Utility library under the “SD” namespace.

The database is accessed through a *Data Access Layer*. If instead each web form that needed to fetch or modify data would contain an appropriate SQL-statement, this would

inevitably lead to poor maintainability due to lack of reuse, and possibly a security threat unless parameterized queries or stored procedures were used. See section 8.3.8 for more on security issues. The Data Access Layer gathers all data access logic in one place and thus makes it easier to maintain and reuse. In addition, table adapters and queries make each query return a typed dataset. See chapter 4 for more information on table adapters.

7.3 Class overview

This section summarizes important classes from the static description. More details on static description is given in section 7.5 below.

7.3.1 Web forms

The web forms define the user interface. They were developed during the interface design described in chapter 6, and later documented in the static description. *Table 7.1* summarizes the main web forms in the design specification.

Table 7.1: Overview – Web forms

Class	**Description**
SDSearch	This page provides an interface for searching for products by entering search terms and specifying optional conditions.
SDProducts	This web form is used to browse lists of products or to display search results from the SDSearch class. It allows the client to add, remove or hide any of the displayed products, and supports sorting, paging, grouping and filtering.
SDCategories	This form provides the link between SDs' categories and the clients' own categories. An SD category can be selected along with a linked own category.
Products	The Products page lists products that have been added to the shop. It is possible to choose to display either hidden, all or just imported products from the drop down box. Grouping category wise is done by a check box and searching is possible. Buttons exist for editing, removing and setting associated products.
ProductDetails	This form allows editing the existing products.
Upload	This form enables uploading, browsing, previewing and choosing images.
Associations	In this form, one or more related products can be selected from a tree view by checking them. There is an option to make the relations symmetric.
Categories	This form enables the user to create and maintain categories.
Pages	This form gathers all the web pages that the client should be able to edit. Pages are grouped by type and have buttons for editing and removing. New custom pages can also be created.

Class	Description
Page	The Page allows editing the contents of a page.
Accounts	This form creates and maintains login accounts. When an account is created, a random password is created and sent to the users email address.

7.3.2 Utility library

The utility library contains common routines used throughout the web forms. *Table 7.2* summarizes the classes in the utility library.

Table 7.2: Overview – Utility library

Class	Description
Authentication	This static class manages login procedures, and checks if the user is authenticated.
CurrentClient	Manages properties for the client associated with the authenticated user account.
CurrentUser	Manages properties for the authenticated user.
DataLookup	This static class contains simple lookup functions for very small datasets.
FileManagement	Contains methods for creating and maintaining the file structure associated with the client, including copying default product images.
SDSearchData	Contains search parameters, and serves as data transport between the classes SDSearch and SDProducts.
Utils	Contains various small utility functions to be reused.

7.4 Final interface design

This is the final interface design made for the MI2. Important designs are presented below with screen shots. Each design is presented with a list of changes from the original design for comparison.

7.4.1 CheckList

The start-up guide can be seen in *Figure 7.2* below along with the MI2 menu to the left. The layout is designed to fit on a screen with resolution 1024x768 pixels, as specified in *NR006: Screen resolution*. This is perhaps the most appreciated page among the users. From a usability perspective, it has been invaluable in helping users through the first time usage. Changes from original:

- Root nodes were enlarged to increase visibility.
- Task formulations were clarified numerous times.

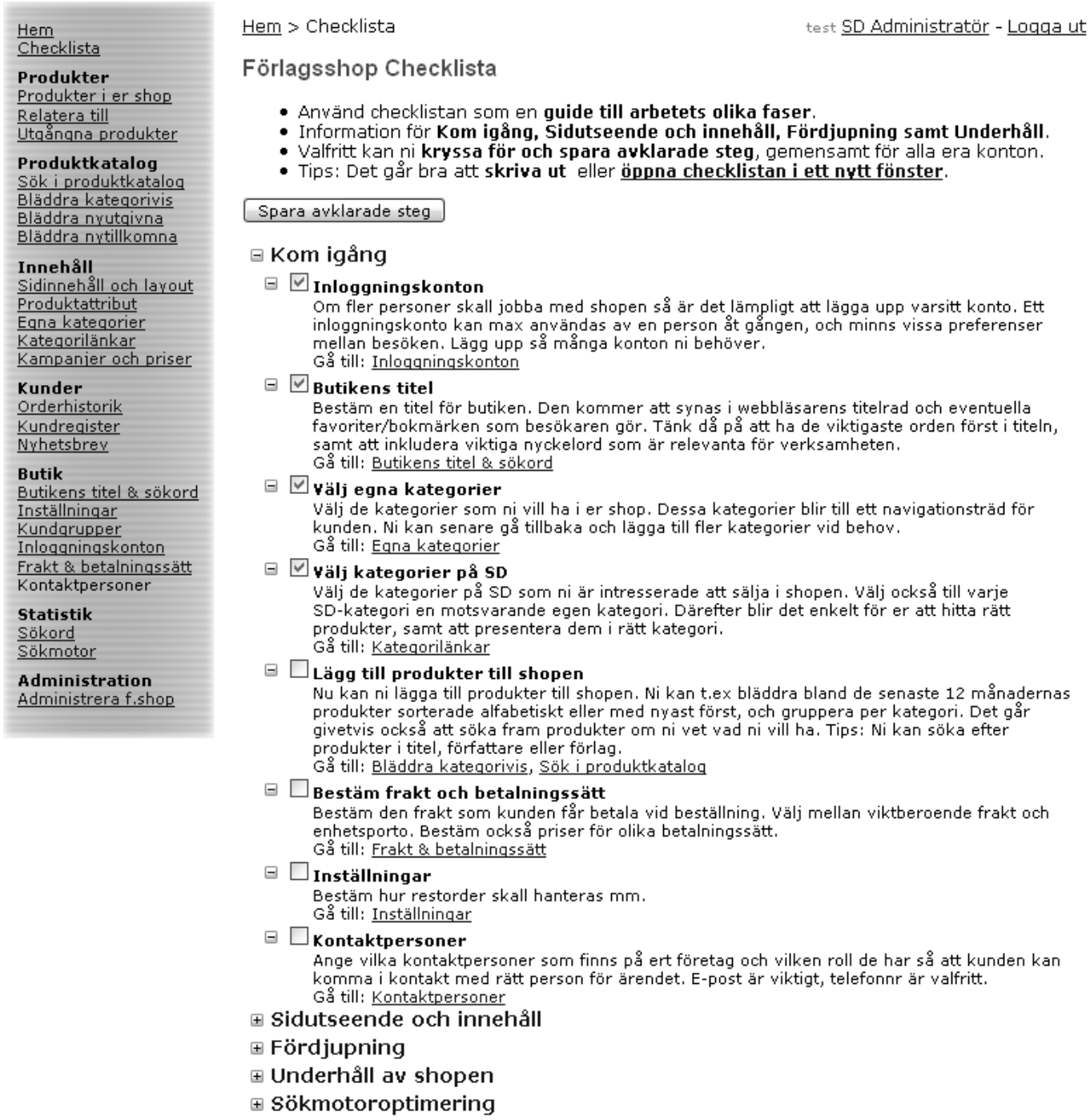

Figure 7.2: Final design – CheckList

7.4.2 Categories

This page shows a typical usage of the GridView component. As you can see in *Figure 7.3*, it allows editing of contents, and there are buttons for all actions on rows. As the page evolved, the following was changed from the original design:

- The page title was changed from “Categories” to “Your categories”.
- A sort button for main categories was added.
- Buttons that perform actions on rows were moved to the actual rows to be consistent with the rest of the design.
- A shortcut was added to “Pages and Layout”, for adding text to category pages.
- It is possible to add a main category and a subcategory simultaneously by using both text boxes.

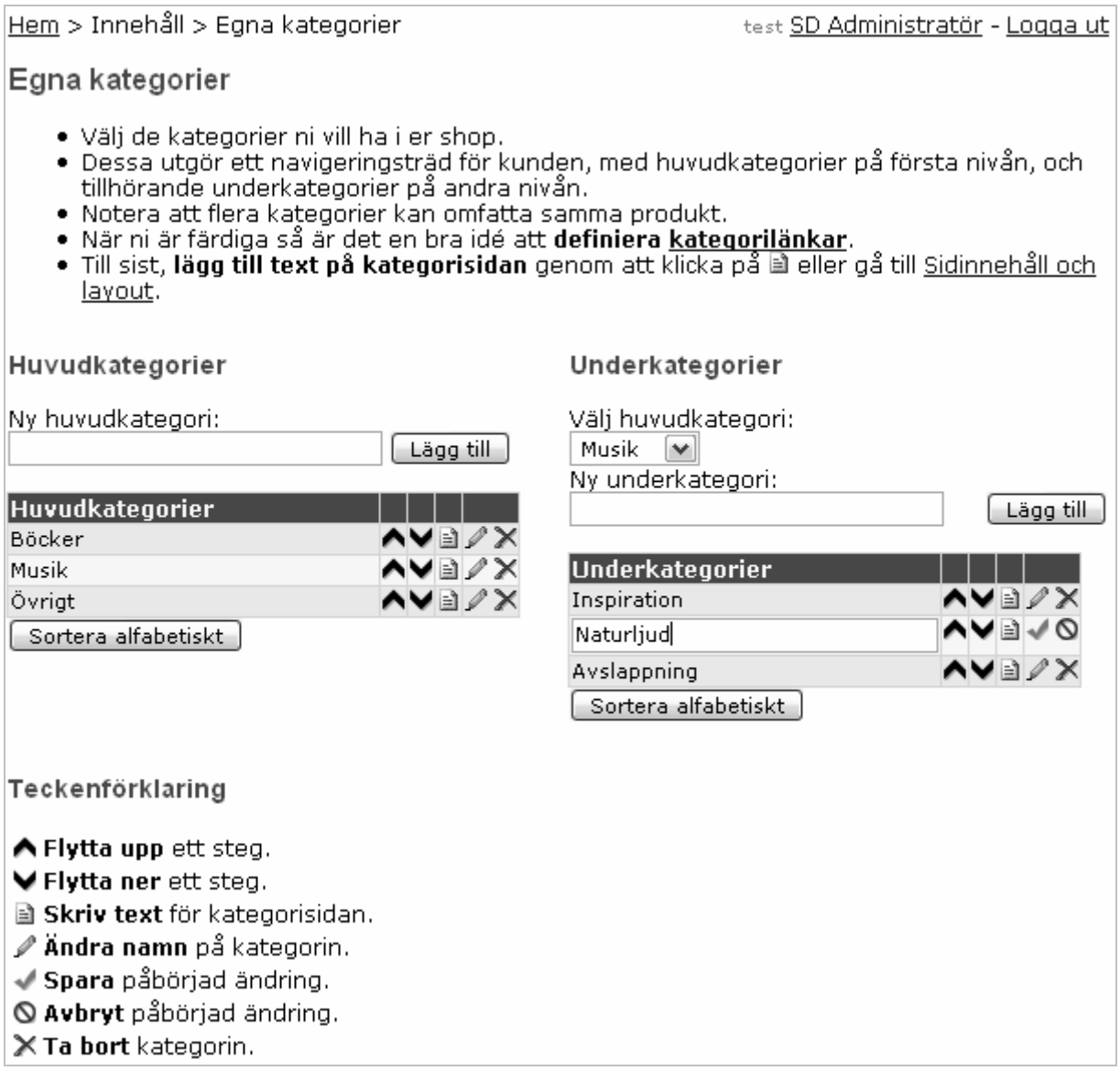

Figure 7.3: Final design – Categories

7.4.3 SDCategories

See *Figure 7.4*. This page did not exist in the original design. Some of the changes during the development were:

- The on-line help has been edited and clarified a number of times to avoid confusion.
- Sub categories are only shown on demand to give better overview and reduce code size.
- If a client category is chosen in the drop down box, but the checkbox was not checked before save, then it is automatically checked to avoid frustrating mistakes.

Figure 7.4: Final design – SDCategories

7.4.4 SDSearch

See *Figure 7.5*. The search page was redesigned after introducing the category links. Changes from the original:

- A filter option was added for searching “In selected categories only”.
- Page title changed to “Search product catalogue”, instead of “Add (search)”.
- The product ID search was changed so that multiple IDs can be entered separated by commas, instead of just one at a time.

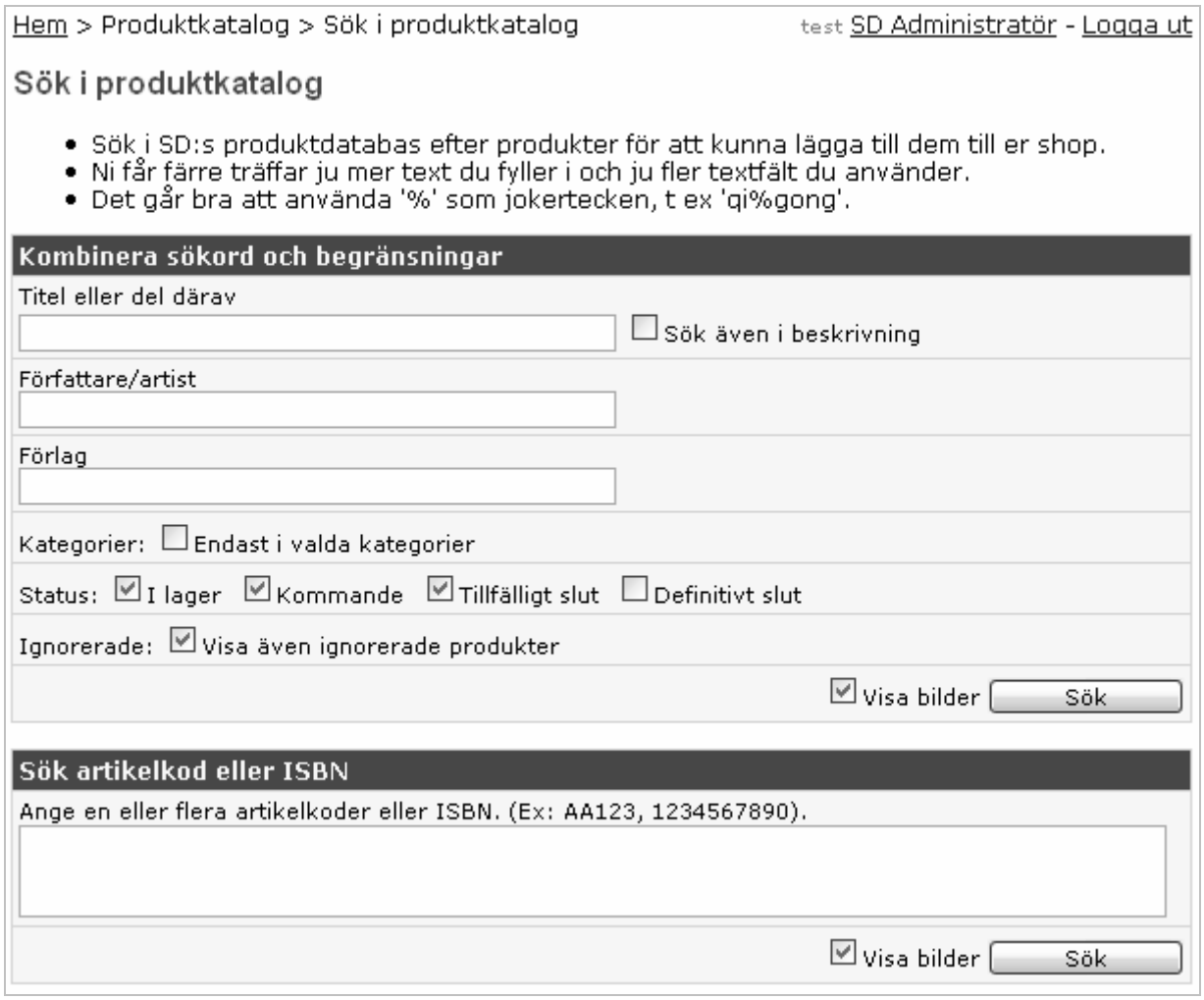

Figure 7.5: Final design – SDSearch

7.4.5 SDProducts

See *Figure 7.6*. Since this page is expected to be one of the most visited pages, a lot of effort has been put into it. The purpose of the page is to find and add products to the shop. Changes from the original:

- A link was added pointing to the search page.
- A checkbox was added to add all products on the page.
- Quick links for browsing were redesigned with emphasis on either showing all products, or showing “In selected categories only”.
- Added online help about how to ignore and unignore a product.
- Added feedback for ignoring/unignoring products.
- Explained why there are no products when browsing “In selected categories only”, but no links are defined.
- Added month-wise group headers when sorting by date added.
- Limited the number of rows per category when grouping and sorting by “category, popularity”.
- When sorting by “category, date added”, the drop down box “published date” is renamed to “registered date”, and the filter works accordingly.
- A popup window shows when the mouse points at an image, displaying a larger image and a full description text.
- The drop down box “Results per page” now has the same values as the corresponding box in the Products page, i.e. 20, 50, 100 and 200 and is persistent over a session.

Figure 7.6: Final design – SDProducts

7.4.6 Associations

On this page, the user can select related products. As you can see in *Figure 7.7*, a TreeView component has been used for organizing products. The original design intended a window to be opened after picking a product. The page was redesigned so that a new window would not have to be opened to select related products. Instead, clicking a title would select a starting product, and from there one would check related products as before, but now in the same window. In addition, media type was added to differentiate products with the same name.

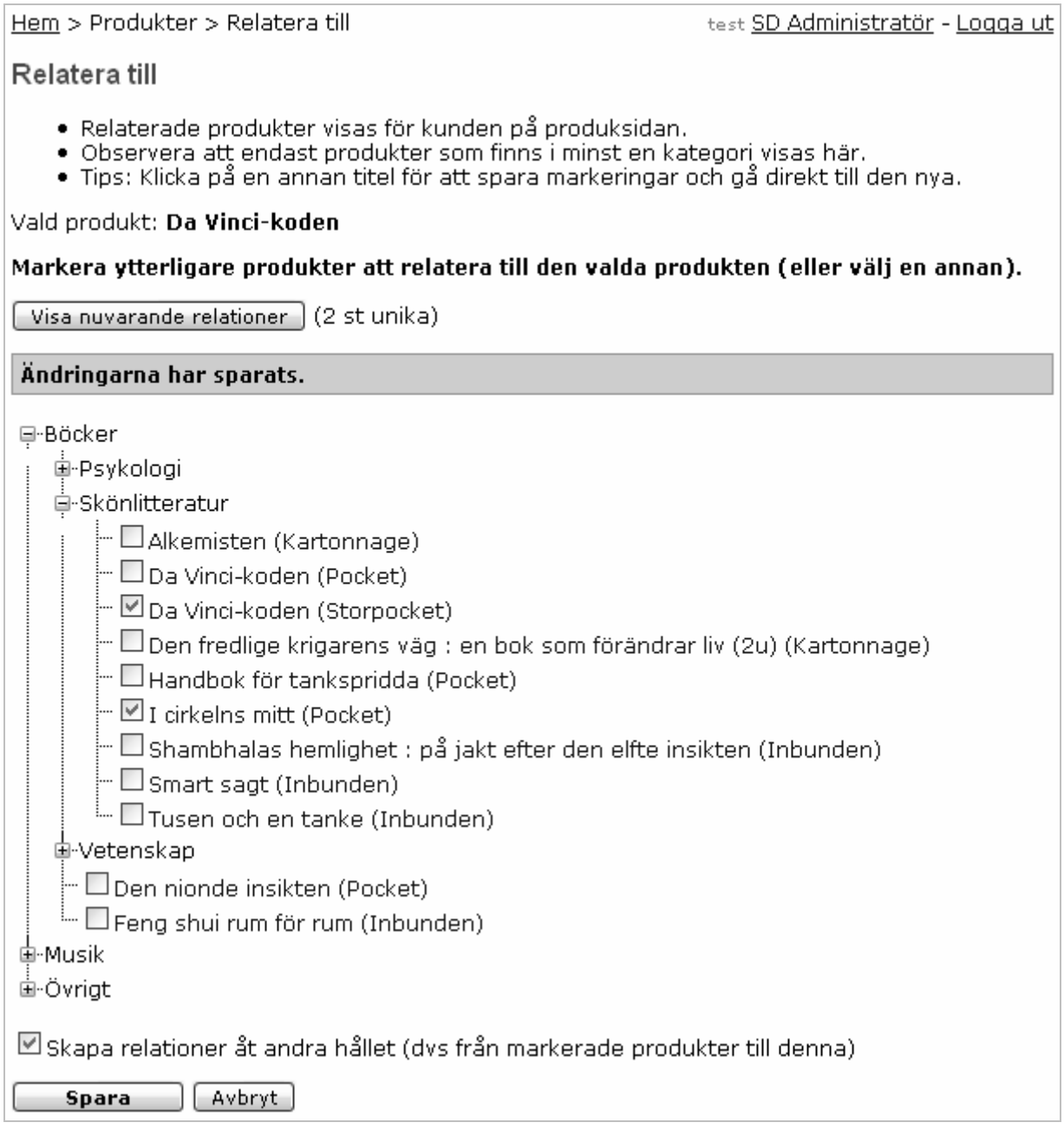

Figure 7.7: Final design – Association

7.4.7 Products

See *Figure 7.8*. This page shows the products in the shop, and has a few buttons for each applicable action. The following has been changed from the original design:

- When the product list was to be implemented, it showed that the original idea with a ASP.NET TreeView was both slow to use and was not flexible since buttons was not supported. Instead, a Repeater was chosen for its flexibility.
- Grouping by category was done with separators titled with the category name.
- Paging was added manually, since the built-in component did not support this directly.
- The drop down box “Results per page” was given the same values as the corresponding box in the SDProducts page, i.e. 20, 50, 100 and 200 and was made persistent over a session.

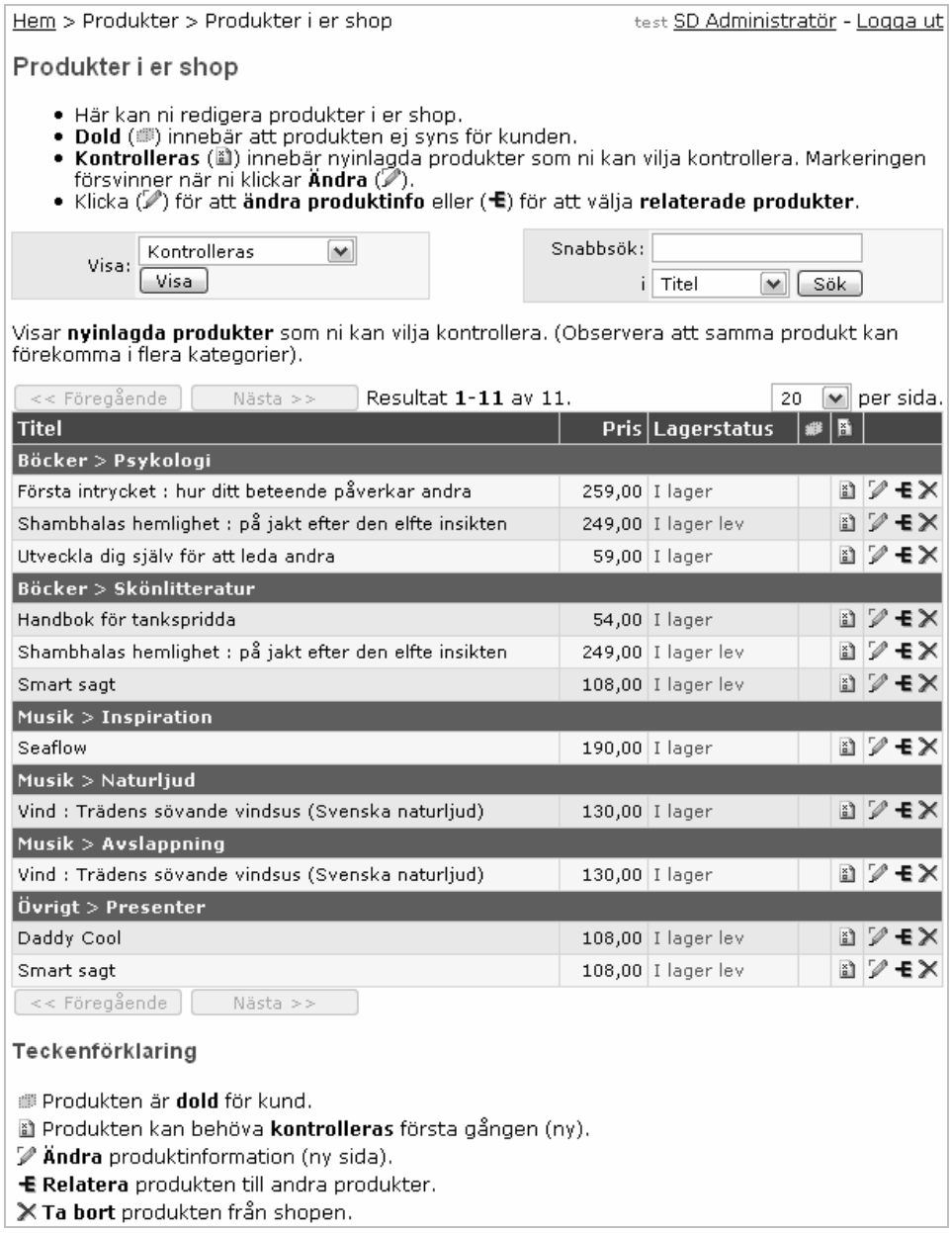

Figure 7.8: Final design – Products

7.4.8 ProductDetails

See *Figure 7.9*. ProductDetails allow editing product details. Changes from the original:

- "Save" and "Cancel" buttons were added at the top of the page, in addition to the ones at the bottom, to make the page consistent with the other pages.
- A button was added for restoring the product to default information.
- When the user clicks "Save" while a row in a GridView is in edit mode, the row is first saved, instead of being discarded without notification as before.

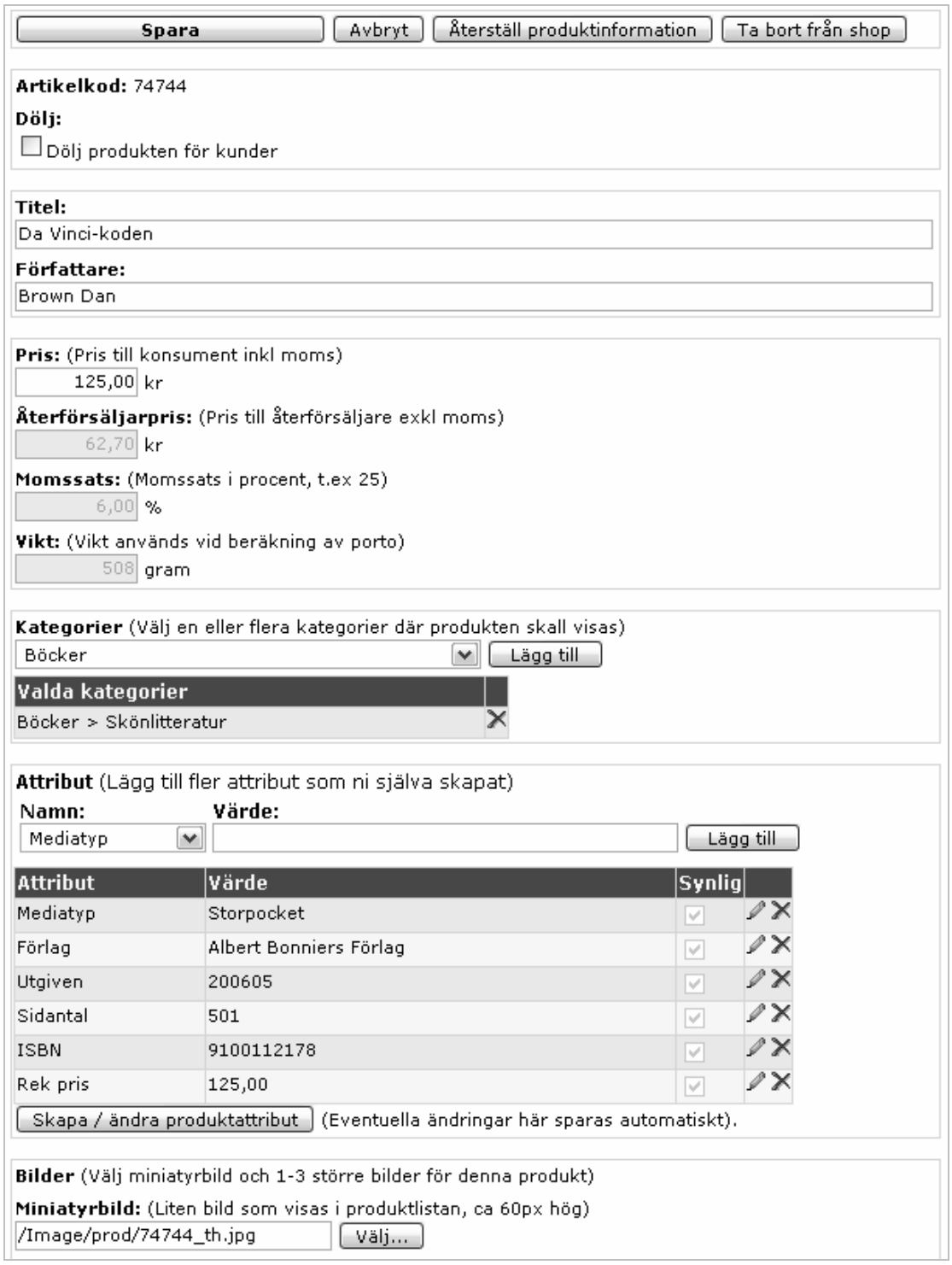

Figure 7.9: Final design – ProductDetails

7.4.9 Upload

See *Figure 7.10*. Upload lets the user upload and select images. Changes from the original:

- The original design intended a TreeView to be used for selecting catalogue. As this would have produced a vast amount of HTML-code, it was replaced by a drop-down box.
- Related buttons “Choose” and “Close window” were grouped together and their caption changed to “Choose image” and “Cancel” respectively to better reflect their purpose.
- The buttons “Select image” etc. were moved above the preview to keep them on the page when viewing oversize images.

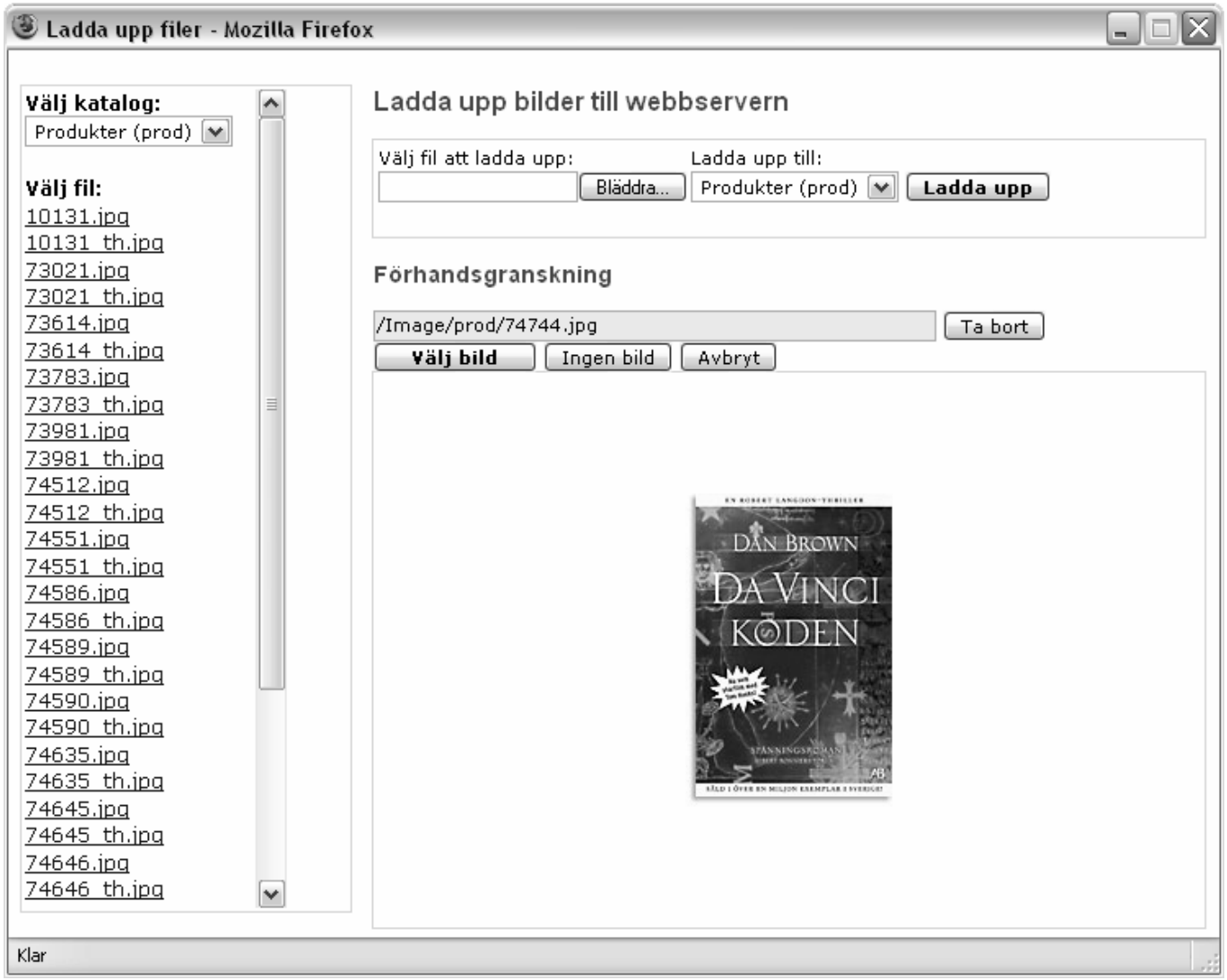

Figure 7.10: Final design – Upload

7.4.10 Pages

This form lists all the pages in the shop that can be edited, as you can see *Figure 7.11*. Very few changes were made to this design from the original:

- Creating a new page was not done as originally thought by actually opening a blank page, but instead there is a form for entering title and filename.

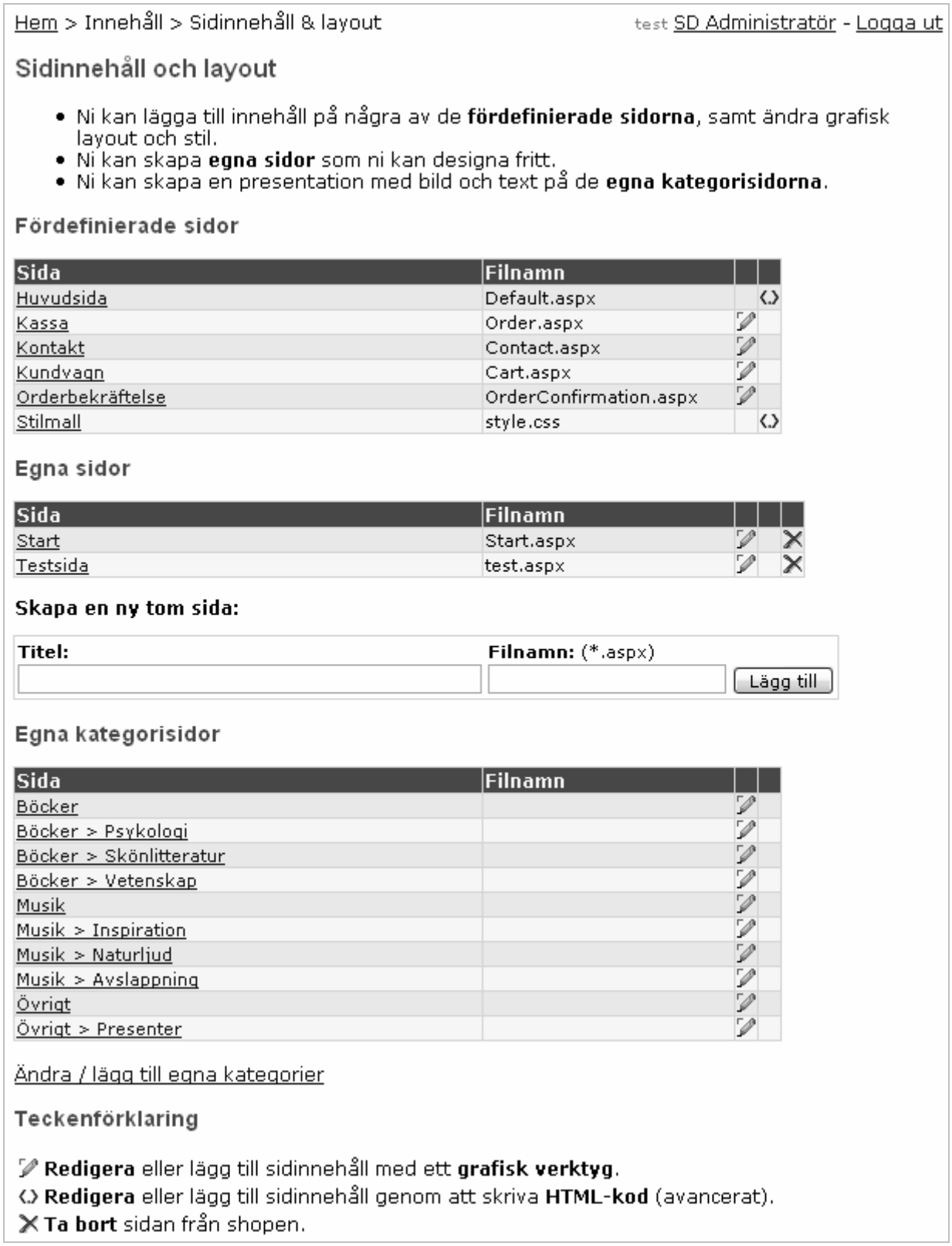

Figure 7.11: Final design – Pages

7.4.11 Page

See *Figure 7.12*. It was uncertain at the time of writing the requirements document whether or not there would be time to implement a full graphical page editor. There was time however, and it proved to be relatively little work involved in integrating the new editor, but it was vital to the usability of the page. Changes from the original:

- A WYSIWYG-editor was implemented with a third part component.
- “Save” and “Cancel” buttons were added at the top of the page for consistency.

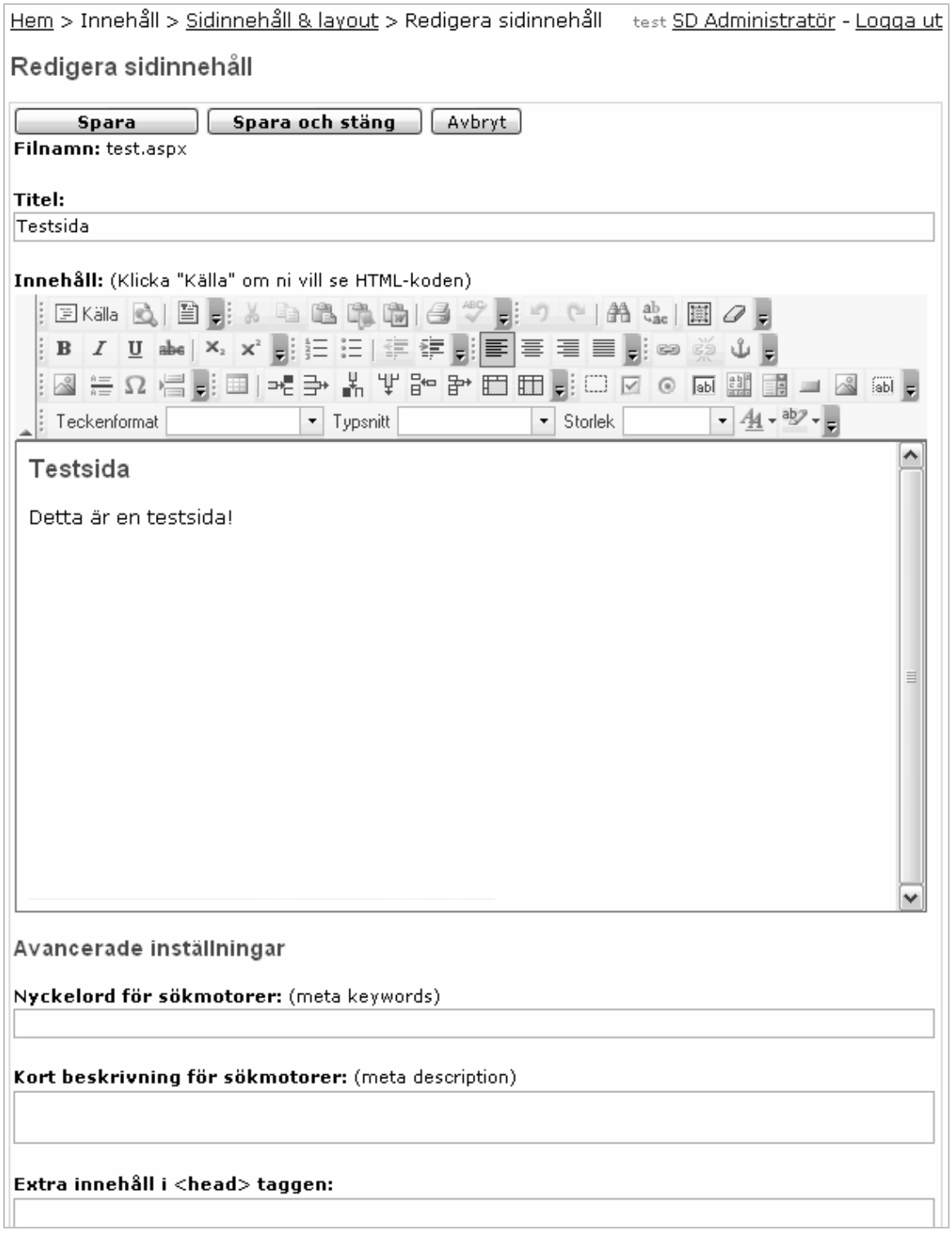

Figure 7.12: Final design – Page

7.4.12 Campaign

The campaign wizard consists of four steps. Steps 2 and 4 are shown in *Figure 7.13* and *Figure 7.14* respectively. The following has been changed from the original:

- "Active" option was removed.
- A page indicator was first added, and then replaced by the options selected so far, titled "Campaign description".
- Made the "Next" button default on the form, instead of "Previous".

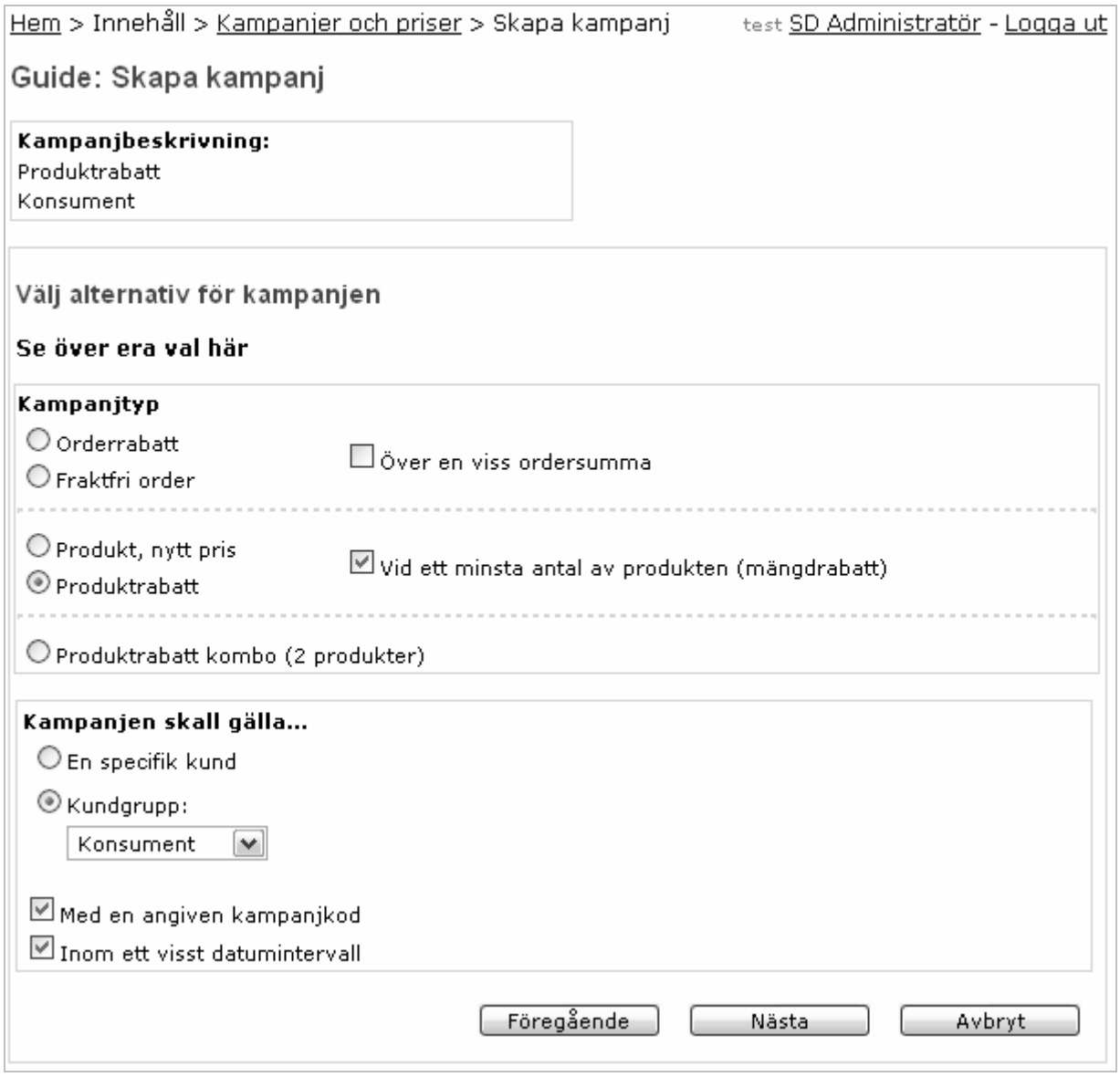

Figure 7.13: Final design – Campaign (alternatives)

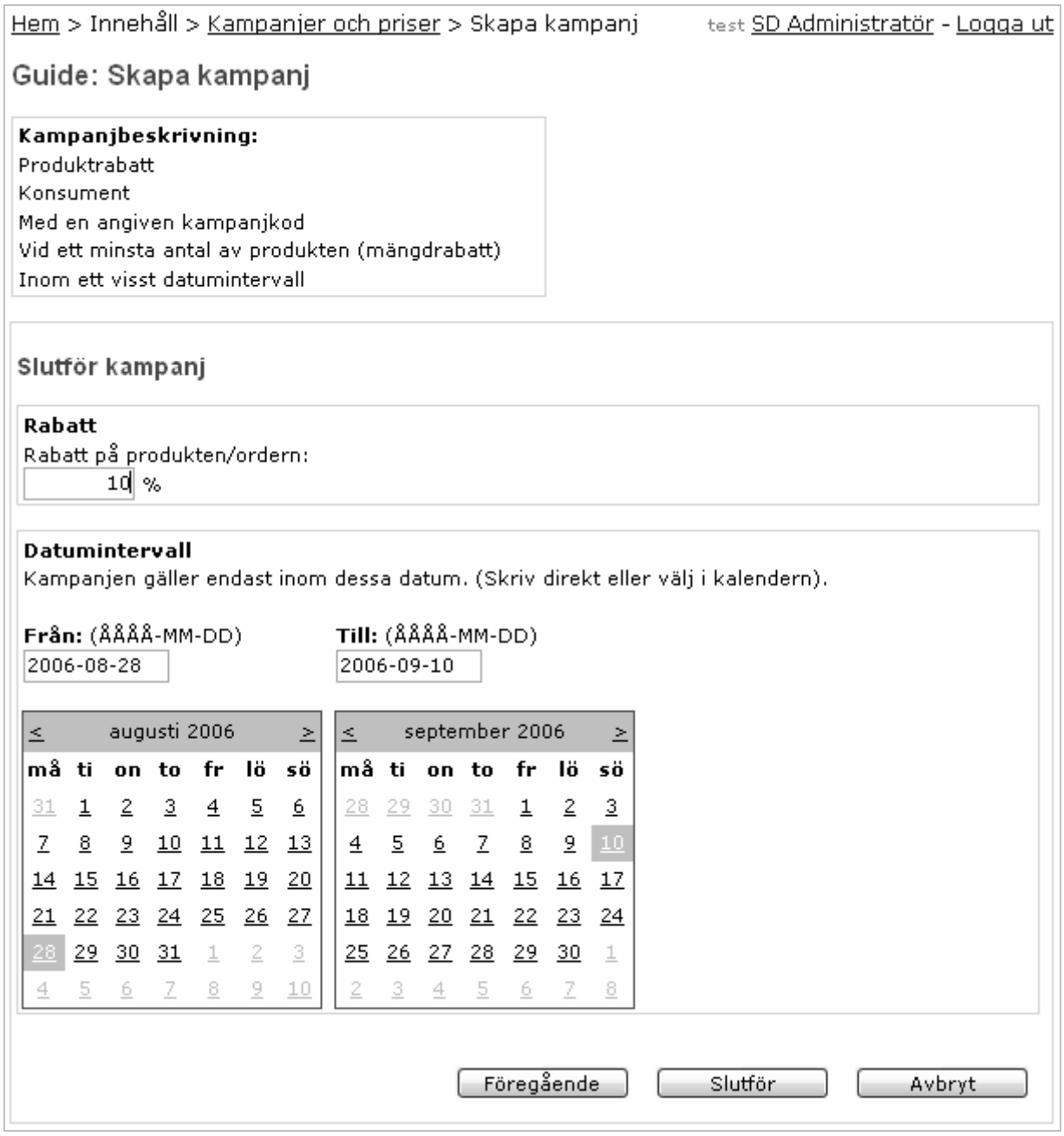

Figure 7.14: Final design – Campaign (completion)

7.5 Static description

The amount of web forms, classes and table adapters involved in the system is huge, and it would be meaningless to describe all of them here. Instead, I refer to the summary in section 7.3 above, and illustrate with an example next of how the SDProducts class is specified.

Example: SDProducts

One of the most important classes of the system is the SDProducts class, illustrated next as an example of a static description.

This web form is used to browse lists of products or to display search results from the SDSearch class. It allows the client to add, remove or hide any of the displayed products,

and supports sorting, paging, grouping and filtering. The objects and methods represented by the class are shown in the class diagram in *Figure 7.15* below.

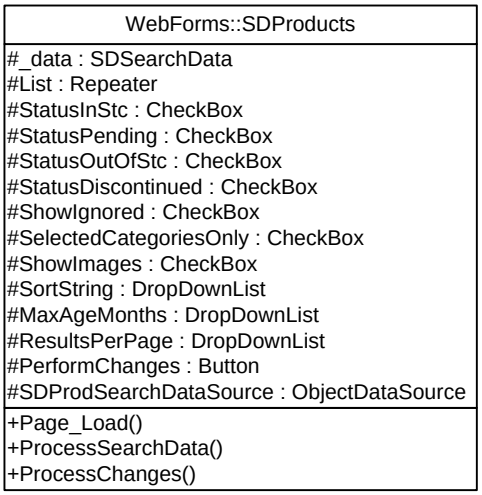

Figure 7.15: Class diagram – SDProducts

Members

These are the members of the SDProducts class.

Table 7.3: Class members – SDProducts

Member	**Description**
_data	This object stores all parameters that affect the search results. Text parameters can be empty, whereby the object represents a filter for browsing products, instead of a search.
List	The main Repeater that displays results in a list format. Its item template contains other controls such as Add, Ignore and Remove checkboxes.
StatusInStc	When checked, filters products in stock.
StatusPending	When checked, filters pending products.
StatusOutOfStc	When checked, filters products out of stock.
StatusDisc	When checked, filters products that have been discontinued.
ShowIgnored	When checked, also shows previously ignored products.
SelectedCategoriesOnly	When checked, filters products matching selected categories in “SDCategories”
ShowImages	When checked, shows images with the results.
SortString	Contains the string that determines sorting and grouping of “List”.
MaxAgeMonths	Contains the maximum age in months as an integer, filtering by published date.

Member	Description
ResultsPerPage	Integer determining the number of results to be displayed in “List”.
PerformChanges	Explicitly saves the changes made to the form.
SDProdSearchDataSource	The main data source for “List”.

Methods

These are the methods of the SDProducts class.

Table 7.4: Class methods – SDProducts

Member	Description
ProcessSearchData	Retrieves search data as a “SDSearchData” object from the previous page. The data is stored in the ViewState. Finally, the data is transferred to the select parameters of “SDProdSearchDataSource”.
ProcessChanges	Processes each item in the form, and depending if the action is to add, ignore or delete a product calls the appropriate database method.

7.6 Dynamic description

The dynamic description focuses on the sequence of interactions between the classes and the client. In addition, some important methods are described. I did not find any particular information in for example Pfleeger [7] regarding how to create a dynamic description. The process was however rather straightforward. Common for almost all descriptions is that the client interacts with the system, and the system reacts with a response. The client can be seen as a web browser or similar in this setup. I used the *sequence diagram* type of the UML (Unified Modelling Language) standard to describe the sequences of interactions between the classes. The boxes represent objects and the vertical lines their life span. The vertical activation bars indicate when a process is active, and the arrows represent messages.

The document has been shortened to fit this report, and only an example is used to illustrate. The same class as in the last example is used – the SDProducts class. This particular design is a bit unusual since it is uncommon for the loosely coupled web forms to interact with each other.

SDSearch and SDProducts

This is a typical scenario; the client makes a search in the product catalogue and adds a few products. See sequence diagram in *Figure 7.16* below. Notice that there is no fixed time between the client’s requests, since he or she needs time to fill in the form.

The client requests the “SDSearch” page. He then submits the search data to “SDProducts”, which calls “ProcessSearchData”. The search data is retrieved as a “SDSearchData” object through the property “GetSearchData” in the previous page. The

data is stored in the ViewState for persistency over the lifespan of “SDProducts”, so that SDProducts can use the search data in subsequent requests. Finally, the data is transferred to the select parameters of the main data source and the result list is displayed.

The client makes changes to some items in the form, and submits it back to “SDProducts”. “ProcessSearchData” gets the same search data from the ViewState this time. “ProcessChanges” is called which processes each item in the form, and – depending if the action is to add, ignore or delete a product – calls the appropriate database method.

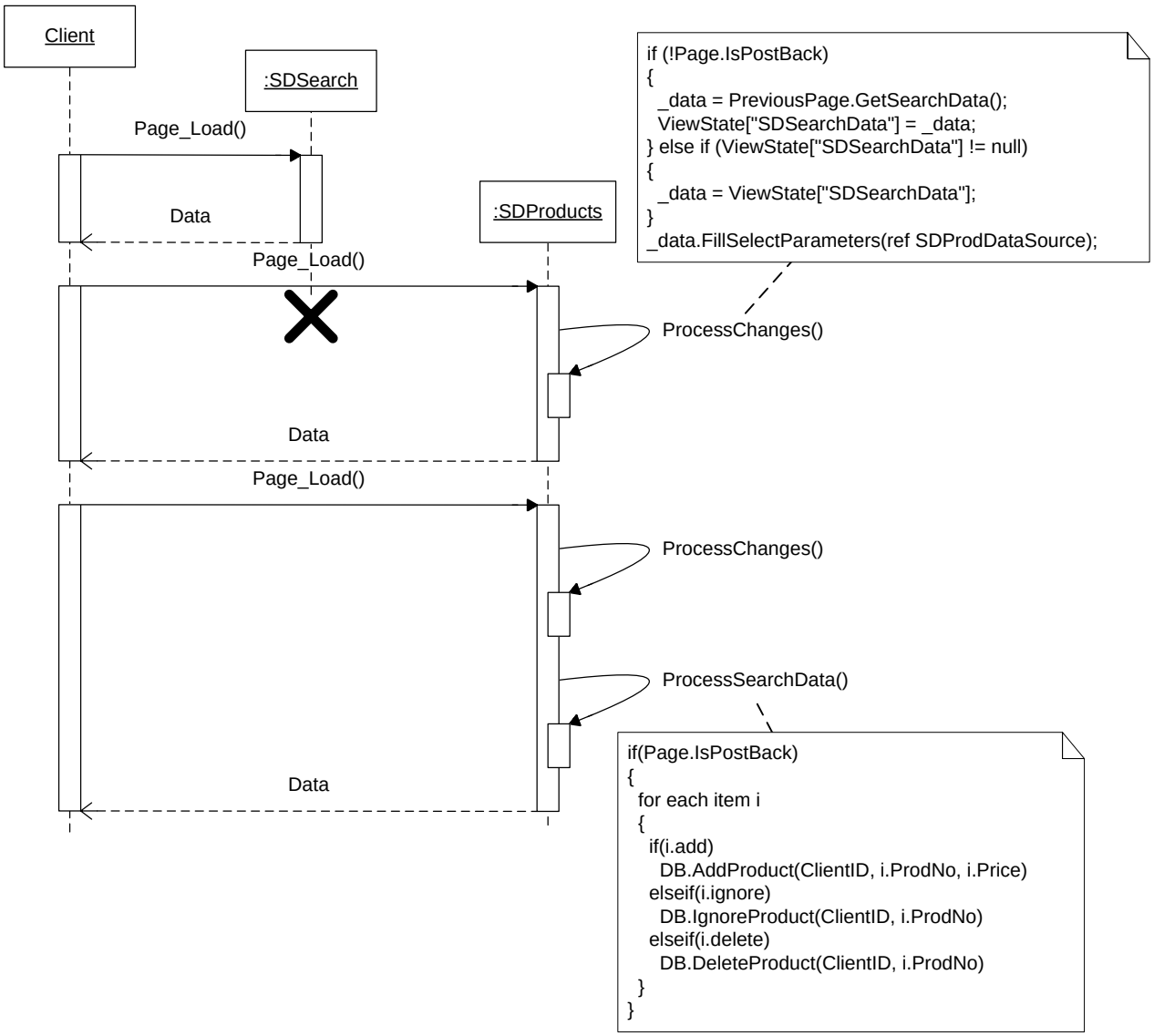

Figure 7.16: Sequence diagram – SDSearch and SDProducts

7.7 Database structure

The database is part of the foundation of the system. Practically all web forms make use of the database in some way. The diagrams used in the system were produced in Enterprise Manager – a tool part of SQL Server 2000. An example is shown below. When creating relationships, one can specify referential integrity constraints and two types of referential integrity actions; namely *cascading updates* and *cascading deletes*. Referential constraints and actions have been heavily used in the database design, in an attempt to make the data

consistent. Their use is further described in section 4.2.7. Some issues with referential constraint actions are discussed in section 8.3.

A thorough review of the requirements for both MI2 and PS2 yielded a list of proposed entities for the database. The entities were then given appropriate attributes, and preliminary relationships were created. As the user interface developed using prototypes, the data model was changed appropriately, like the addition of entities and attributes. In addition, improvements that were not apparent at the start were made later in the process.

There are many ways to design a database for the same application. As an example, the table “Cat” was previously represented by two tables; one for main categories and one for sub categories. Then I realized that the structure of the two was almost identical, and spawned the current design with a recursive relation. Several similar improvements were made in order to get a database structure that is consistent, robust, and simplifies coding and data maintenance.

During the development, I chose not to normalize the relations to normal forms beyond the first as described in chapter 3. A clear example of this is the table “Prod”, where Title and Author could be placed in other relations. However, a major issue in this system is that the data needs to be easy to maintain in order to keep the user interface simple, which is why I allow breaking the “rules” in cases like this. If for example “Author” were extracted to another relation, then the client would have to enter authors separately, and then choose from some list. It would greatly complicate the design if this were to be done for all such cases.

An example diagram from the system is included below to show how it may look.

7.7.1 Database design example: Products

Following is a description of tables and relations concerning products. The database diagram is shown in *Figure 7.17*, and the tables are briefly described in *Table 7.5*.

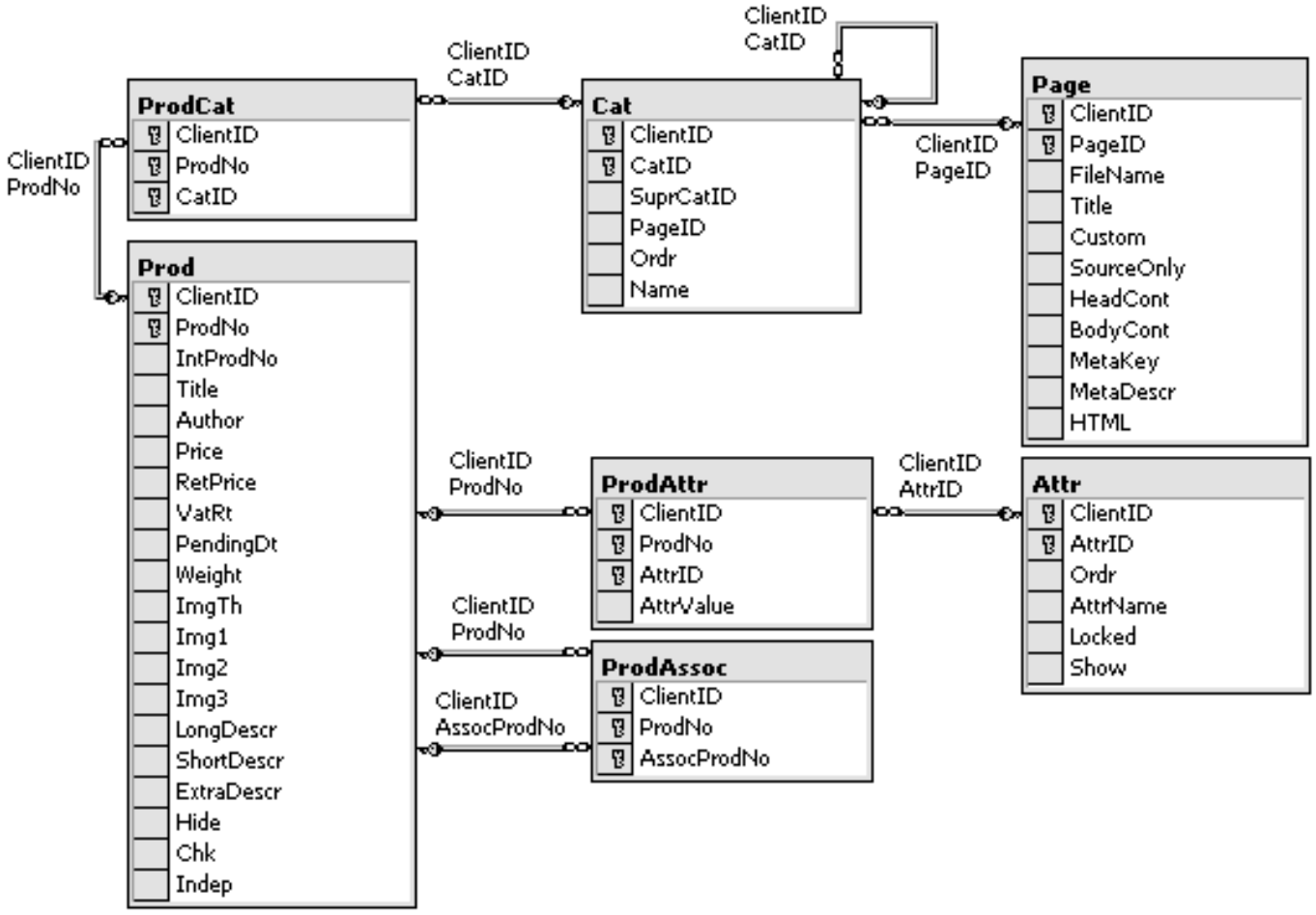

Figure 7.17: Relationsships – Products

Table 7.5: Database tables – Products

Table	Description
Prod	Prod represents a product in the front-end. Default information is duplicated from SDProd since the client should be able to modify it at will.
Cat	Products are grouped into a hierarchy of categories defined by Cat. Instead of creating one table for each level, we define a reflexive relation between the keys CatID and SuprCatID. We let SuprCatID be zero at the top level. Each category can also have a page associated with it.
ProdCat	ProdCat is the key entity to the many-to-many relation between Prod and Cat, so that every product can belong to several categories.
ProdAssoc	ProdAssoc enables a product to be associated to zero or more other products in a many-to-many relation.
Attr	Attr defines an array of attributes names for each client to be used with products. There is a default set of attributes from the start, which are locked from deletion. In addition to that, the client can add his own attribute names.
ProdAttr	ProdAttr lets the client assign one or more attributes to each product, pairing attributes from Attr with values in ProdAttr.

8 Discussion

This chapter looks at and discusses different aspects of the project, such as the system itself, the methods used and some implementation issues.

8.1 System

The MI2 system has been designed to replace the existing back-end. However, the PS2 front-end still remains to be developed. Without it, MI2 is rather useless. MI2 has been demonstrated to the manager at SD, and he was very pleased with the result. PS2 is currently on the drawing board, and is scheduled for delivery in four months from the time of writing. Thus, the system has met the anticipations of SD, and the complete integrated system is expected to be a great success. Several customers have already committed to the new PS2 system, even before it has been designed.

My own estimate of time required for implementation was initially one month. However, this proved to be a very optimistic estimation. The actual time taken for implementation and testing was close to three months. One major factor in the delay was unforeseen problems, such as the required customization of the standard controls discussed in section 8.3. Most of the issues required additional time for research and experimentation, which slowed down the development.

I am personally satisfied with the final design and implementation of MI2. It seems to me that the user interface is consistent, powerful and in many ways self-explanatory. Short and concise on-line help is provided for those in need of extra support, and the start-up guide provided was well appreciated by the users.

The implementation is clean with code clearly separated from user interface design, thanks to the programming model of ASP.NET. ASP.NET proved to have the power required to create the system, although many “tricks” had to be learned to fulfil all requirements. I will therefore conclude that ASP.NET was a good choice for this application. SQL Server 2000, which provided the database platform for the system, also worked sufficiently.

8.2 Method

8.2.1 Usability Engineering

The requirements gathering activity was done in place of the task analysis that is normally a part of the Usability Engineering activities, because we already had a good understanding of the users’ tasks. I do believe however, that it still would have been worthwhile to do the extra formal work of documenting it as the task analysis prescribes, because it would have given us a better overview and perspective on the tasks.

The iterative development approach attempted to combine Usability Engineering tasks with requirements elicitation through throw-away prototyping. It is evident that new requirements were discovered in the process, at the same time as usability issues were discovered, thereby fulfilling the purposes of both paradigms.

During the interface design, the phases design, testing and evaluation were iterated four times. An additional fifth redesign stage was performed at the end. The first three

iterations were tested by the discount usability testing method, and evaluated using the User Satisfaction Questionnaire. The average results are illustrated in *Figure 8.1* below.

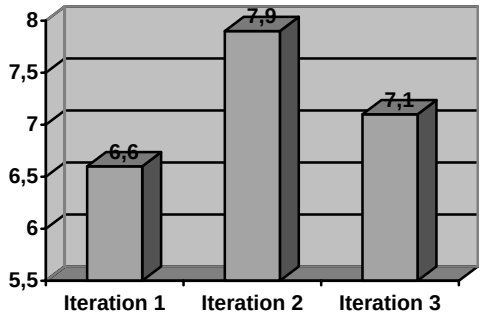

Figure 8.1: Result comparison – User Satisfaction Questionnaire

I should point out that the statistical basis is very small due to the few participants involved, so we should be careful to draw any conclusions from the results.

We notice that the second iteration yielded higher results than the first, which suggests an improvement to the design. The third iteration however, gave lower scores than the second iteration. This can be explained by the fact that the first two iterations used a low-fidelity paper prototype, while the third used a screen prototype. Therefore, users may have had lower expectations than if it had been a screen prototype. In addition, many more questions were answered in the third iteration than in the previous two, since they lacked relevance for the paper prototype. The additional relevant questions gave the users a chance to nuance their response, which may have lowered the average result. It would have been interesting to evaluate the final system for comparison.

The two iterations with paper prototyping were valuable because there were some major redesigns in the beginning, and the throwaway prototypes were rather quick to develop. Looking back however, at some point I could have introduced a software prototyping tool, which likely develops prototypes at a similar speed, but with higher fidelity. After observing the users interacting with the paper-prototype, it was evident that some difficulties were due to the lack of detail. A software prototyping tool could have maintained the rapid development, while increasing detail compared to the paper prototyping, and thereby improved the usefulness of the usability testing sessions.

If more users had been involved each iteration, I believe we subsequently would have gotten more response and more accurate questionnaire scores. The general guidelines for discount usability testing suggest three to six participants. In this project, one to two users participated in each test. Still, much feedback was gained, and many problems were identified despite the low number of participants.

The stop criteria for the usability testing stated that the phases "…should be iterated at least three times, and finish when a fix point is reached, where the improvement in subjective satisfaction of the users is less than 10% between the iterations". This is an alternative way to state when to stop, which is relatively easy to use. A typical approach otherwise is to set limits for usability measures, such as those listed in chapter 3. The

second method allows more detailed measures to be used, but also requires experience in writing achievable requirements.

We now look at some other possible methods of usability testing, as listed in chapter 3.

- *Competitive usability testing* would require a similar system to compare with. Since the type of service that is being developed is rather unique, we have yet to find such a similar system.
- *Universal usability testing* is hardly relevant, since the users of the system are few and known, and a questionnaire has shown that their configurations are similar.
- *Field tests* and *remote usability testing* are possible methods that could be offered to additional users. Due to lack of resources and the fact that users still cannot see the result of their work (because PS2 is not finished), these methods have been ruled out for the moment. Still, some sort of field test is likely to occur, once the integrated system is deployed.

Two expert reviews in the form of heuristic evaluations were performed during iteration four. A lack of qualified personnel limited us to two such reviews, but clearly more testers would have identified more problems. The results of this type of evaluation were substantially different from those of the usability testing done previously. Primarily, the results of the expert evaluations focused on consistency and on details important to usability, while usability testing clearly gave results according to the problems experienced.

Surely, other expert review methods could also have been successful, but the heuristic evaluation was appropriate under the circumstances.

One issue open for discussion, is the sequencing of usability testing and expert reviews. We could progress either by interleaving the two methods, or by combining them at each iteration. The former would yield diverse feedback, while the latter would be more time-consuming, but would combine the advantages of both methods.

Due to the diverse results from expert reviews and usability testing, I believe it is advantageous to use both methods at some stage in a project, no matter when it is done.

8.2.2 Software engineering

We now look at some of the Software Engineering methods used during the development.

The requirements gathering process involved examining the previous system, requirements from clients, doing interviews and a questionnaire survey. All these methods contributed to the final set of requirements. For the process to be successful, I believe it is important to mix what designers think the system needs, with what the users think the system needs. We must also recognize that user interface designers generally have a tendency to be attached to the first design, even though another design actually is better. One principle I know of is to develop multiple design suggestions, but always drop the first one.

Developing the technical design was done at the same time as the user interface design. The components placed on the user interface determined the static structure to some extent. Additional requirements were discovered during design prototyping.

Some interesting implementation issues are discussed in section 8.3.

Testing was not done formally in this project. It was investigated if it was possible to do unit testing on ASP.NET 2.0 web forms. The “NUnitASP” project came up, but it did not support version 2.0 of the framework. Since the GridView was extensively used in MI2, but could not be tested with “NUnitASP”, this solution was abandoned. Microsoft ships a version of the development environment called Team System 2005, which is said to support unit testing. However, this was outside of our budget.

8.3 Implementation issues

The report did not go into depth on the technical aspects of the development. However, I would like to discuss some interesting aspects of programming with ASP.NET. Doing advanced programming with ASP.NET gave some complications, and usually needed some research and experimentation. Here are some of the issues that were discovered during implementation.

8.3.1 Static variable scope

The scope of static variables is *across the server*, not for the *life of a call* to a particular page or during a particular session. If you set the value once, all subsequent calls made to that same ASP server process will have access to this value.

I discovered this the hard way when attempting to implement a *Singleton* style class, see [5, p. 127]. The class was meant to keep track of client-specific details, such as client ID and user name. As soon as another client logged in, the Singleton was reused, and the previous details were lost, forcing the first client the same details as the second.

The problem was that frequently used details stored in the database would cause unnecessary database access during the execution of a single page, since for example the client id is needed by several data sources.

The solution I found, although not very beautiful, was to put frequently used data in session variables, encapsulated by the classes ClientInfo and UserInfo. When referencing data not in the session, data will be fetched once and cached for the duration of the page execution.

8.3.2 Grouping with GridView

Initially, the product list in “Prod” was a GridView. Then I realised that grouping was needed. If for example you are grouping by category, then each product that starts a new category should be preceded by a header, stating the category name. I searched the MSDN and other forums, and found a couple of code samples. The ideas circled around the time of data binding an item, trying to insert a group header dynamically into the dataset to which the GridView was bound. I was successful to some extent, but as soon as I changed page, or deleted a record there would be an incomprehensible error message. In the end, I abandoned the GridView, and converted to a Repeater instead, to which I had eventually found a working solution.

8.3.3 Grouping with Repeater

The solution I found for making group headers with a Repeater worked every time. It was not beautiful however, since it mixes some imperative programming with the otherwise purely declarative syntax of ASP.NET. Each item is described in an item template, which outputs a table row. I begin the template with another row for the group header. The trick is to only display it when the header changes. I solved this using an instance of the help class SD.Utils, setting visibility of the group row to the value of the function "IsDifferentFromPrevious", which returns true if and only if the supplied value (Object) is different from the previous value.

8.3.4 Hiding columns in a GridView

The MI2 system was designed for both SAs and clients. A client has fewer options than the SA does, so sometimes we need to show only a subset of columns in a GridView. Hiding columns in a GridView causes them to disappear from the dataset. Suppose there is a hidden field that is part of the update method. When making updates directly from the GridView, this field is not included, and thus by default passed as a null value to the update method. Most of the time this is not desired, since the original value would be erased. Instead, one would like the hidden field to be passed as the original value, but this proved to be just too difficult.

The solution was to create separate update methods for different column sets. This of course creates redundant procedures, and complicates maintenance.

8.3.5 Verifying input and deletion in GridView

As mentioned in chapter 3, the `ASP:GridView` component supports editing and deleting items. Usually, one will want to verify the input before allowing an update to be performed. ASP.NET provides a set of controls known as validators that adds excellent client and server side support for validation of input. However, there is no immediate way to make use of these controls in the GridView. Instead, one must write individual item templates for each column that is to be verified.

If one wants the user to confirm deletion of a row, a similar method is used with templates combined with client script that is activated with the delete button.

8.3.6 Container name length

One of the challenges when creating web pages is keeping the download size small. Container components are those that encapsulate other components, prefixing their own id to all of the contained components' ids.

Suppose a particular Repeater has an id with 14 characters, and each item contains 20 other components such as divs, labels, checkboxes etc, which is not unusual. Each of the 20 components would then have a prefix with 14+1 characters (one for the separator). Multiply the number of components with say 50 rows, which makes for 1 kb per character of the containers id. Therefore, the 14-character name makes for 15 kb of extra code, just for naming.

If instead the container name is shortened down to four characters, the additional code is reduced to 5 kb, making a substantial difference of 10 kb.

This is the reason I chose to make container names, especially data controls, as short as about 3-4 characters. The disadvantage is code that is a bit harder to understand.

Other common containers that suffered from the same or similar problem were the GridView, TreeView and validators.

8.3.7 Multiple paths with referential integrity actions

The two referential integrity actions *cascading updates* and *cascading deletes* (see section 4.2.7) were used in as many relationships as possible in the database design. The advantage is that the database is kept consistent. If for example, a product is deleted from the table “Prod” in *Figure 7.17*, then any associated attributes and categories in tables “ProdAttr” and “ProdCat” will automatically be deleted to preserve referential integrity.

The problem occurs when there are multiple cascading paths. Multiple paths can exist if there is more than one relationship leading in to some foreign key. According to the rules of SQL Server 2000, there can be at most one such path, which means that only one path can have cascading deletes active. Thus with multiple paths, in at least one case there is a possible deletion that will not cascade. Instead, it will cause the referential integrity constraints to activate, which in practice throws an error.

To prevent errors, something has to be done. So far, I have found two solutions. The first method uses *triggers*, mentioned in section 4.2.7, with which a script can be invoked on a delete action and perform a delete in related tables, assuming this can be done before the error is thrown. The second method works by using stored procedures for actions such as deletion, whereby additional delete actions can be performed before the main delete action. The latter is the one used in this project.

8.3.8 Security considerations

Security in web applications could be dedicated a whole book, but I will only scratch the surface. Even with the built-in protection that ASP.NET provides, mentioned in chapter 3, there are still vulnerabilities that needs attention.

One such issue is known as SQL injection. A system vulnerable to such an attack enables the hacker to execute SQL code on the server. The purpose of the code can be to cause havoc in the system, gain access to protected pages, or to harvest information by exploiting information handed out by certain types of built-in error messages.

An application receives data from so-called GET or POST requests. A typical GET request can look like `Page.aspx?id=12`. The application reads the data and typically uses it to fetch some data from a database. Assume the system is vulnerable to SQL Injection and simply creates an SQL statement as a string, and uses the value “id” from the query string like so:

```
“SELECT * FROM page WHERE id=” + id;
```

The hacker can then replace the numeric value with a string[7]:

```
Page.aspx?id=1;DELETE * FROM page
```

When assembled to the final SQL statement, it will have changed the meaning of it:

```
“SELECT * FROM page WHERE id=1;DELETE * FROM page”;
```

This new statement has the side effect that it deletes all the rows from the table.

We should therefore not trust data that is received as request data. Various techniques can be used to protect from this type of attack.

1. Keep the permission level as low as possible for the web application, like disallowing DROP and DELETE statements.
2. If the data is supposed to be numeric, make sure that it actually is numeric before attempting to use it.
3. If the data is a string, set a maximum length of the string to cause difficulties to the attacker.
4. Use either parameterized queries or stored procedures (mentioned in section 4.2.7) instead of a plain text command. The nature of parameters in SQL makes this type of attack very difficult, if not impossible.
5. Adopt custom error messages that hide the details of the error, to prevent information harvesting.

The MI2 uses stored procedures exclusively.

[7] Spaces in a query string are usually represented as “%20”, but are shown as spaces here for clarity.

9 Conclusion

The user interface was developed by a Usability Engineering approach involving iterating the three stages of redesign, usability testing and evaluation. The user interface was designed as a paper mockup in the first two iterations, and was thereafter implemented before further testing. This approach caused the usability testing to halt for a period. A better way may have been to use a software prototyping tool, which in addition could have gained responses that are more detailed. Nevertheless, the method was sufficient and allowed for usability testing at several stages during the development.

The usability testing involved only one to two users, but still identified many usability problems, preparing for the next redesign. Expert reviews in the form of heuristic evaluations were used at the end of the project, and while only two “experts” participated, the method still produced a long list of suggested improvements.

The result of the usability work was a user interface that met the expectations of the company. The developed system is pending integration with its corresponding front-end, and will thereafter be deployed in a production environment.

References

[1] Anderson, Francis, Homer, Howard, Sussman, Watson (2001). “Proffesional ASP.NET”. Wrox Press.

[2] Carlshamre (2001). “A Usability Perspective on Requirements Engineering – From Methodology to Product Development”. Institute of Technology, Linköpings Universitet.

[3] Conrad, Dengler, Francis, Glynn, Harvey, Hollis, Ramachandran, Schenken, Short, Ullman (2000). “Introducing .NET”. Wrox Press.

[4] Elmasri, Navathe (2003). “Fundamentals of Database Systems”. 3rd edition. Pearson Higher Education.

[5] Gamma, Helm, Johnsson, Vlissides (2002). ”Design Patterns - Elements of Reusable Object-Oriented Software”. Addison-Wesley.

[6] Kroenke (2004). “Database Processing - Fundamentals, Design and Implementation”. 9th ed. Prentice-Hall.

[7] Pfleeger (2001). “Software Engineering - Theory and Practice“. Prentice-Hall.

[8] Shneiderman, Plaisant (2005). “Designing the User Interface - Strategies for Effective Human-Computer Interaction”. 4th ed. Addison Wesley.

Web references

[9] Microsoft. “.NET Framework Developer Center: Technology Overview”. <http://msdn.microsoft.com/netframework/technologyinfo/overview/default.aspx> (2006-09-05).

[10] Microsoft. “Get Started : The Official Microsoft ASP.NET 2.0 Site : English”. <http://www.asp.net/getstarted/default.aspx?tabid=61> (2006-09-05).

[11] Microsoft. “System.Web.UI.WebControls Namespace”. <http://msdn2.microsoft.com/en-us/library/system.web.ui.webcontrols.aspx> (2006-09-06)

[12] University of Maryland. “Questionnaire for User Interaction Satisfaction”. <http://lap.umd.edu/QUIS> (2006-04-14)

Appendix A – Questionnaires

A.1 User questionnaire

The user questionnaire was handed out during the requirements gathering to collect information about the clients. The answers are summarized here.

The numbers in square brackets ([ ]) indicates the number of answers for that alternative.

Age
[ ] Up to 20 years
[ ] 21 to 25 years
[ ] 26 to 30 years
[4] 31 to 40 years
[3] 41 to 50 years
[1] Above 50 years

Sex
[4] Male
[4] Female

Part 1: Environment

1.1 To which of the following web browsers do you have access?
[8] Internet Explorer
[ ] Mozilla Firefox

1.2 What type of Internet connection do you have where you work?
[1] Modem 56K
[1] Modem ISDN
[4] Modem ADSL
[2] Cable

1.3 What screen resolution (roughly) do you have where you work?
[ ] 640x480 pixels
[ ] 800x600 pixels
[1] 1024x768 pixels
[2] 1280x960 pixels
[ ] 1600x1050 pixels or larger
[5] Don’t know

Part 2: Computer experience
(The numbers after “>” indicates the answer from each user)

2.1 Do you feel you have much computer experience?
(Disagree) 1 2 3 4 5 6 7 (Agree)
> 3 – 4 – 4 – 5 – 6 – 6 – 6 – 7

2.2 Do you enjoy learning new things about how programs and systems work? (Disagree) 1 2 3 4 5 6 7 (Agree)
> 3 – 3 – 4 – 4 – 6 – 7 – 7 – 7

2.3 Which of the following programs and systems are you familiar with and have used?
[6] Word processor (like MS Word)
[5] Spreadsheet (like MS Excel)
[2] Database (like MS Access, FileMaker)
[6] Web editor (like MS FrontPage, Dreamweaver and GoLive etc)

2.4 Do you feel confident in Microsoft Excel or similar spreadsheet?
(Disagree) 1 2 3 4 5 6 7 (Agree)
> 1 – 2 – 2 – 2 – 5 – 5 – 6 - 7

2.5 Do you feel confident in Microsoft Access, FileMaker or similar database program?
(Disagree) 1 2 3 4 5 6 7 (Agree)
> 1 – 1 – 1 – 2 – 2 – 2 – 2 - 5

2.6 Do you feel confident in web tools?
(Disagree) 1 2 3 4 5 6 7 (Agree)
> 1 – 1 – 1 – 2 – 3 – 4 – 5 - 6

2.7 Do you feel confident in HTML (language for web pages)?
(Disagree) 1 2 3 4 5 6 7 (Agree)
> 1 – 1 – 1 – 1 – 1 – 2 – 3 - 6

Part 3: Merchant Interface 2.0

3.1 Administration
[6] Option to create an independent shop, where orders go directly to the Publisher

[7] Possibility to se an order history with customer info and order details
[ ] Other: _____

3.2 Performance
[6] System should respond fast across a modem connection
[4] Several users can use the system at the same time
[ ] Other: _____

3.3 Usability
[6] There are short help texts at every field in the interface
[4] There is an on-line manual
[ ] Other: _____

3.4 Language
[4] Possibility to create a shop with pages and product info in English
[4] Possibility for the visitor to see prices in different currencies
[ ] Other: _____

3.5 Newsletters
[5] Possibility to send newsletters in plain text to the visitors that have opted for this.
[5] Possibility to send newsletters with text and images (HTML) to the visitors that have opted for this.
[5] Possibility to make selective send outs, for example to members only
[6] Possibility to track the response from send outs, i.e. number of clicks
[6] Possibility to track how many orders that a send out led to
[ ] Other: _____

3.6 Statistics
[7] Possibility to see statistics over what your visitors look for in your shop
[5] Possibility to see statistics over which search engine that referred to your shop
[6] Possibility to see statistics over which search terms that lead the visitor to your shop
[ ] Other: _____

3.7 Products

[6] Possibility to add products to the shop online
[5] Possibility to add products via a spreadsheet
[6] Possibility to see new products from SD, which can be added with a single click
[5] Possibility to choose related products that are shown near a certain product
[5] Products by the same author/artist are shown near a certain product
[6] Possibility to display a product in more than two categories
[ ] Other: _____

3.8 Campaigns
[5] Possibility to create a time limited campaign on individual products
[5] Possibility to create general time limited campaigns with a total order discount
[5] Possibility to have hidden campaigns where the visitor must enter a campaign code
[ ] Other: _____

3.9 Customer login
[5] Enable the customer to create an account that remembers the customers' details
[ ] Other: _____

3.10 Member prices
[5] Possibility to have separate prices for members
[2] Possibility for the visitor to purchase a one-year membership
[ ] Other: _____

3.11 Retailer login
[5] Possibility to choose which products that should be accessible for retailers
[6] Possibility to let the visitor order as a retailer (approved by SD) for retailer price
[5] Possibility to register a retailer and assign them a login account
[ ] Other: _____

3.12 Page contents

[5] Possibility for the customer to write a review of the product (implies that you would have to oversee and possibly modify posts)
[6] Possibility to create a description of the author with a picture by the product
[4] Possibility to choose sorting order for products, for example title or publish date
[ ] Other: _____

3.13 Front page
[5] Possibility to show product news
[5] Possibility to show bestsellers
[5] Possibility to show campaign products
[ ] Other: _____

3.14 Other
If you have suggestions for things that you would find useful in the Merchant Interface, please write them here
> ...

A.2 User satisfaction questionnaire

This is the questionnaire for user feedback that was used to evaluate the user interface after each discount usability test.

User satisfaction questionnaire

```
Identification number: _______
Age: ____
Gender: __ male __ female

Please circle the numbers which most appropriately reflect your impressions about
using this computer system. NA = Not Applicable.

Part 3: Overall user reactions
3.1 Overall reactions to the system:  terrible              wonderful
                                        1  2  3  4  5  6  7  8  9      NA
3.2                                   frustrating          satisfying
                                        1  2  3  4  5  6  7  8  9      NA
3.3                                   dull                 stimulating
                                        1  2  3  4  5  6  7  8  9      NA
3.4                                   difficult            easy
                                        1  2  3  4  5  6  7  8  9      NA
3.5                                   inadequate power     adequate power
                                        1  2  3  4  5  6  7  8  9      NA
3.6                                   rigid                flexible
                                        1  2  3  4  5  6  7  8  9      NA

Please write your comments about your overall reaction here:
```

Part 4: Screen

4.1 Characters on the computer screen	hard to read	1	2	3	4	5	6	7	8	9	easy to read NA
4.2 Highlighting on the screen	unhelpful	1	2	3	4	5	6	7	8	9	helpful NA
4.3 Screen layouts were helpful	never	1	2	3	4	5	6	7	8	9	always NA
4.4 Sequence of screens	confusing	1	2	3	4	5	6	7	8	9	clear NA
4.4.1 Next screen in a sequence	unpredictable	1	2	3	4	5	6	7	8	9	predictable NA
4.4.2 Going back to the previous screen	impossible	1	2	3	4	5	6	7	8	9	easy NA
4.4.3 Progression of work related tasks	confusing	1	2	3	4	5	6	7	8	9	clearly marked NA

Please write your comments about the screens here:

_____

_____

Part 5: Terminology and System Information

5.1 Use of terminology throughout system	inconsistent	1	2	3	4	5	6	7	8	9	consistent NA
5.1.2 Work related terminology	inconsistent	1	2	3	4	5	6	7	8	9	consistent NA
5.1.3 Computer terminology	inconsistent	1	2	3	4	5	6	7	8	9	consistent NA
5.2 Terminology relates well to the work you are doing	never	1	2	3	4	5	6	7	8	9	always NA
5.3 Messages which appear on screen	inconsistent	1	2	3	4	5	6	7	8	9	consistent NA
5.4 Messages which appear on screen	confusing	1	2	3	4	5	6	7	8	9	clear NA
5.4.1 Instructions for commands or functions	confusing	1	2	3	4	5	6	7	8	9	clear NA
5.4.2 Instructions for correcting errors	confusing	1	2	3	4	5	6	7	8	9	clear NA
5.5 Computer keeps you informed about what it is doing	never	1	2	3	4	5	6	7	8	9	always NA
5.5.2 Performing an operation leads to a predictable result	never	1	2	3	4	5	6	7	8	9	always NA
5.5.4 Length of delay between operations	unacceptable	1	2	3	4	5	6	7	8	9	acceptable NA
5.6 Error messages	unhelpful	1	2	3	4	5	6	7	8	9	helpful NA
5.6.1 Error messages clarify the problem	never	1	2	3	4	5	6	7	8	9	always NA
5.6.2 Phrasing of error messages	unpleasant										pleasant

											1 2 3 4 5 6 7 8 9 NA

Please write your comments about terminology and system information here:

_____

_____

Part 6: Learning

6.1 Learning to operate the system	difficult									easy	
		1	2	3	4	5	6	7	8	9	NA
6.1.1 Getting started	difficult									easy	
		1	2	3	4	5	6	7	8	9	NA
6.1.2 Learning advanced features	difficult									easy	
		1	2	3	4	5	6	7	8	9	NA
6.1.3 Time to learn to use the system	difficult									easy	
		1	2	3	4	5	6	7	8	9	NA
6.2 Exploration of features by trial and error	discouraging									encouraging	
		1	2	3	4	5	6	7	8	9	NA
6.2.1 Exploration of features	risky									safe	
		1	2	3	4	5	6	7	8	9	NA
6.2.2 Discovering new features	difficult									easy	
		1	2	3	4	5	6	7	8	9	NA
6.3 Remembering names and use of commands	difficult									easy	
		1	2	3	4	5	6	7	8	9	NA
6.4 Tasks can be performed in a straightforward manner	never									always	
		1	2	3	4	5	6	7	8	9	NA
6.4.1 Number of steps per task	too many									just right	
		1	2	3	4	5	6	7	8	9	NA
6.4.2 Steps to complete a task follow a logical sequence	never									always	
		1	2	3	4	5	6	7	8	9	NA
6.4.3 Feedback on the completion of sequence of steps	unclear									clear	
		1	2	3	4	5	6	7	8	9	NA

Please write your comments about learning here:

_____

_____

Part 7: System Capabilities

7.1 System speed	too slow									fast enough	
		1	2	3	4	5	6	7	8	9	NA
7.2 The system is reliable	never									always	
		1	2	3	4	5	6	7	8	9	NA
7.3 Complexity of common tasks	unacceptable									acceptable	
		1	2	3	4	5	6	7	8	9	NA
7.4 Correcting your mistakes	difficult									easy	
		1	2	3	4	5	6	7	8	9	NA

Please write your comments about system capabilities here:

_____

Part 8: On-line help

8.1 Information from the checklist	confusing								clear	
	1	2	3	4	5	6	7	8	9	NA
8.2 Amount of help given	inadequate								adequate	
	1	2	3	4	5	6	7	8	9	NA
8.3 On-line help	useless								helpful	
	1	2	3	4	5	6	7	8	9	NA

Please write your comments about on-line help here:

Part 9: Navigation

9.1 The system lets you know where on the site you are	never								always	
	1	2	3	4	5	6	7	8	9	NA
9.2 Link names are	unpredictable								predictable	
	1	2	3	4	5	6	7	8	9	NA

Please write your comments about navigation here:

Appendix B – Eight golden rules of interface design

The following principles are described by Shneiderman & Plaisant [8, p.74] as the eight golden rules of interface design:

1. **Strive for consistency**. “Consistent sequences of actions should be required in similar situations; identical terminology should be used in prompts, menus and help screens; and consistent color, layout, capitalization, fonts and so on should be employed throughout”.
2. **Cater to universal usability**. “Recognize the needs of diverse users and design for plasticity, facilitating transformation of content. Novice-expert differences, age ranges, disabilities, and technology diversity each enrich the spectrum of requirements that guides design. Adding features for novices, such as explanations, and features for experts, such as shortcuts and faster pacing, can enrich the interface design and improve perceived system quality. “
3. **Offer informative feedback**. “For every user action, there should be system feedback. “
4. **Design dialogues to yield closure**. “Sequences of actions should be organized into groups with a beginning, middle, and end. Informative feedback at the completion of a group of actions gives operators the satisfaction of accomplishment, a sense of relief, the signal to drop contingency plans for their minds, and a signal to prepare for the next group of actions. “
5. **Prevent errors**. “As much as possible, design the system such that users cannot make serious errors; for example gray out menu items that are not appropriate and do not allow alphabetic characters in numeric entry fields. If a user makes an error, the interface should detect the error and offer simple, constructive, and specific instructions for recovery.”
6. **Permit easy reversal of actions**. “As much as possible, actions should be reversible. This feature relieves anxiety, since the user knows that errors can be undone, thus encouraging exploration of unfamiliar options. “
7. **Support internal locus of control**. “Experienced operators strongly desire the sense that they are in charge of the interface and that the interface responds to their actions. Surprising interface actions, tedious sequences of data entries, inability to obtain or difficulty in obtaining necessary information, and inability to produce the action desired all build anxiety and dissatisfaction. “
8. **Reduce short-term memory load**. “The limitation of human information processing in short-term memory requires that displays be kept simple, multiple-page displays be consolidated, window-motion frequency be reduced, and sufficient training time be allotted for codes, mnemonics, and sequences of actions. “

Appendix C – Requirements specification

This chapter describes the requirements specification that was produced for the MI2 project. The requirements definition and requirements specification are combined into this single document.

C.1 Document structure and definitions

This section will give you an overview of how the requirements are structured.

Document structure

The requirements are divided into functional and non-functional. Functional requirements are further divided into their category of operation such as products or statistics. Non-functional requirements are divided into *performance*, *usability*, and *environmental* requirements.

Requirement structure

Each requirement has a unique identifier that is written at the beginning of the title, such as FR001. Functional requirements describe input, processing and output of the system. Each requirement is assigned a priority as mentioned below.

Priority levels

Priority is divided into three levels. High priority means that the requirement is to be implemented. Normal priority means that it is to be implemented if all high priority requirements are implemented and there is time left. Low priority means that it will be considered for the next release.

C.2 Users of the system

Two kinds of users will be using the system; the clients and the super administrators (SAs). Most of the requirements regard the client, but the super administrator may be asked by the client to perform some of his tasks, whereby the super administrator would be involved in all parts of the system.

Client

The client is between 30 and 50 years old, and either male or female. He regards himself as being reasonably experienced with computers, and does not mind or actually enjoys learning new systems. He might have some experience of using a web design tool, but has very little or no experience in writing HTML. He is not used to database tools but may have some experience in spreadsheet tools. At work, he uses Internet Explorer and probably has an ADSL internet connection.

Super Administrator

The super administrator is between 24 and 45 years old and works at SD. He has extensive knowledge of both the system at hand and handling the type of information involved, the reason being that he has worked with similar systems before, and daily handles this type of information. He has medium knowledge of writing HTML, and daily uses database

programs and spreadsheets. He uses Internet Explorer and has a high-speed internet connection.

C.3 Functional requirements

Product related requirements

FR001: Finding new products

The client shall be able to add a product that exists in SD's database. In the first step, the client can either search or look at lists of new or upcoming titles. Search can be done by title, author or product id. The displayed result list can then be sorted by title, author, date or category. The list can also be grouped by product category, and filtered by selected categories (FR023). If there are many results, they should be divided into several pages. Any products that are already in the clients' database should have an indication of this fact.

Input

The client either enters search terms or chooses "New products" / "Upcoming products".

Processing

The system performs a search in SD's product database for the keywords or conditions given by the user.

Output

A list of products with relevant information such as title, author, type and options to add (FR002) or ignore (FR003) a product.

Priority

High

FR002: Adding products

After finding one or more products (FR001), the client can choose to import them into the system by selecting them, and clicking a button. Default product information such as title, author and text is transferred.

Input

Products selected for import.

Processing

The selected products are imported to the clients' database and the information is adapted. Product images are copied.

Output

Confirmation message.

Priority

High

FR023: Choosing subscriber categories

The client can choose a set of SD categories that will be used to find and filter new products. For each chosen SD category, the client chooses a category in the shop that new products will automatically receive.

Input

A set of selected categories and associated shop categories.

Processing

The selection is verified and saved.

Output

Confirmation message.

Priority

Normal

FR024: Subscribing to changes

The client can choose to subscribe to changes in the shop. For a start, the letter should contain a list of new products in the categories of the clients' choosing (FR023). If any products have been removed from the shop due to being discontinued, they too should be listed. The letter is sent to the clients' email address at regular intervals.

Input

Regular timed event.

Processing

The letter is assembled.

Output

E-mail sent directly to the client.

Priority

Low

FR003: Hiding new products

After finding one or more products (FR001), the client can choose to hide it from the result list. A hidden product is displayed again only on demand, and is useful to simplify the view from unnecessary clutter.

Input

Products selected for hiding.

Processing

The selected products are remembered so that they can be excluded in future result listings.

Output

The same search results without the hidden products.

Priority

Normal

FR004: Finding existing products

The client should be able to perform operations such as removing or editing a product in his database. A product is first selected either by searching, by browsing a list of all products, or by browsing category wise. Uncategorized or just imported products that need attention are given extra emphasis. Search can be done by title, author or product id. The displayed result list can then be sorted by title, author or date. If there are many results, they should be divided into several pages.

Input

Search terms if searching.

Processing

The system performs a search in SD's product database for the keywords or conditions given by the user.

Output

A list of products with relevant information such as title, author, type along with options to remove or edit a product (FR005), or select related products (FR007).

Priority

High

FR005: Editing existing products

The client should be able to edit information of each product in his database. After finding an existing product (FR004) and choosing edit, the product information can be edited, such as title, text and price. Custom categories and attribute name-value-pairs can be applied. Images can be uploaded and selected (FR006).

Input

Form contents.

Processing

The information is validated before updating the database.

Output

Confirmation message.

Priority

High

FR006: Uploading and selecting images

The client can open a window for selecting images for different purposes. Images are organized in selectable pre-defined catalogues such as "products", "buttons", "logos" and "other". When the client selects a catalogue, the content is displayed in an alphabetical scrollable list with image name only. It should be simple to preview the image. The client

can choose to upload an image to the selected catalogue. An image can be selected, whereby the window closes.

Input

Initial catalogue.

Processing

Processing includes changing catalogue, retrieving catalogue contents and receiving an uploaded file.

Output

List catalogue contents, previewing selected or uploaded images. Returns selected file name.

Priority

Normal

FR007: Selecting related products

The client should be able to select (or associate) related products that are displayed in a list to the customer on the front-end product page. After selecting a primary product (FR004), the client selects each product that should be related to the primary product. Before committing, the client can choose if the relations should be symmetric. Symmetric means the relations should exist both ways.

Input

Selected primary product.

Processing

On start, already existing relations are displayed. On exit, the selected relations are saved into the database.

Output

None.

Priority

Normal

FR008: Creating custom product attributes

Apart from pre-defined attribute names, the client should be able to create new attributes. If localization (FR022) is applied then each attribute name should exist once for each language, and the interface should support the client to perform this translation.

Input

Name of new attribute.

Processing

The name is validated, checked for duplicates and then added to existing attributes.

Output

The updated attribute list.

Priority

High

FR009: Creating categories

The client should be able to create categories. Main categories are first created. For each main category, several sub categories can be created. Each sub category is then tied in 1:N relation. A product can belong to several categories of both kinds. Categories can be ordered for display. If localization (FR022) is applied then each category name should exist once for each language, and the interface should support the client to perform this translation.

Input

Name of category.

Processing

The category is validated, checked for duplicates and then added to existing categories.

Output

The updated category lists.

Priority

High

Customer related requirements

FR010: Customer categories

The client should be able to define customer categories. A customer category has a name and defines if he is given ordinary or retail prices, and if he will be able to search the whole of SD's product register (extended search) in addition to the selected products or not. One category is marked as default and is automatically assigned to new customers. The client can later change the customers' category (FR011).

Input

Category name and settings.

Processing

The data is validated. Category names are checked for duplicates. Only one category can be the default.

Output

The data is updated and displayed.

Priority

High

FR011: Create customer accounts

The client should be able to create customer accounts. Besides common customer details, the client can assign a customer category (FR010) and an internal customer id that is only visible to the client. A password can be automatically generated. The login information is sent to the customer by email.

Input

Customer details and settings.

Processing

The input is validated, and saved. A password may be generated at random.

Output

Login info is sent to the customer by email. A confirmation message is shown.

Priority

High

FR012: Send newsletter

The client should be able to send newsletters to customers who have agreed to receive them. The newsletter can be of only text, or HTML. The HTML version should be editable in a WYSIWYG-editor. Pictures can be uploaded and used in the newsletter (FR006). There should always be a link for unsubscribing at the bottom of each letter.

Input

Newsletter contents, consisting of text or HTML.

Processing

The content is stored in a history list along with recipient count. The list of recipients is filtered for duplicates.

Output

The newsletter is sent to all recipients.

Priority

High: Text only.

Normal: HTML.

Price related requirements

FR013: Creating campaigns

The client should be able to create campaigns. A campaign has a description, a date interval and either gives a discount in percent of order sum or makes the order freight free. There can be conditions for the campaign such as minimum product sum, campaign code and up to two products that need to be on the order for the campaign to be valid. A campaign can be active (visible) or inactive (invisible).

Input

Campaign data.

Processing

The data is validated and inserted into the database.

Output

An updated list of campaigns.

Priority

High

FR014: Creating price exceptions

The client can define price exceptions bound to specific products that provide a description and either a new price or a product discount in percent. Conditions can be applied in a variety of combinations choosing from minimum quantity, date interval, campaign code, customer category and id.

Input

A combination of parameters describing the price exception.

Processing

The data is validated and inserted into the database. Check especially that the condition combination is valid. Either price or discount needs to be specified. Description is optional.

Output

An updated list of price exceptions.

Priority

High

Administration requirements

FR015: Creating multiple client logins

The client should be able to create several login accounts that can be used to enter the MI2. Initially there is one client account that cannot be removed. There is also a hidden SA account used by SD staff.

Input

Login name, password, full name and email address.

Processing

Data is validated. Login name must be unique to the current MI2 (not globally unique).

Output

An updated list of login accounts.

Priority

Normal

FR016: Starting up the MI2

The SA must be able to start up an MI2 instance for a particular client.

Input

Actor number, directory name and email address of new client account.

Processing

A process is initiated where relevant template files are copied to a new client directory and the database is prepared for the new client. A login account for the client is created with login name Admin and a random password.

Output

An email is sent to the client with login details, and the operation is confirmed to the SA.

Priority

Normal

FR017: Editing page content

The client should be able to edit page contents. There are four types of pages; the main page, predefined pages, custom pages and category pages. The main page contains layout information and is edited in HTML mode. All other pages can be edited in either HTML or WYSIWYG mode. Predefined pages include checkout and order confirmation. Custom pages are for example menu, header, start and company description. Category pages have one page for each category, which defines extra contents that is placed before the actual products. Images can be uploaded and used on any page (FR006). If localization (FR022) is implemented then there needs to be one page for each language.

Input

The client clicks a page to start editing.

Processing

The page content is saved in a database.

Output

An updated page list.

Priority

High: HTML editing.

Normal: WYSIWYG editing.

Statistics related requirements

FR018: Viewing common search engine terms

The client should be able to view recent search terms typed by users to find the web site. The selection is based on a limited number of days. Only the last 90 days are kept in

database. The client can choose to sort the list by keyword (with or without keyword grouping) or by date.

Input

The user clicks a link, and may choose sort order and grouping.

Processing

Search terms are grouped and sorted.

Output

A list of ordered search terms.

Priority

Normal

FR019: Viewing local search terms

The client should be able to view recent search terms typed by users to find items in the web shop. The selection is based on a limited number of days. Only the last 90 days are kept in database. The client can choose to sort the list by keyword (with or without grouping) or by date.

Input

The user clicks a link, and may choose sort order and grouping.

Processing

Search terms are grouped and sorted.

Output

A list of ordered search terms.

Priority

Normal

FR020: Newsletter response tracking

The system should ensure that all links in a HTML newsletter are tracked for clicks. The client can later view statistics on number of receivers and number of clicks on each link in the newsletter.

Input

Newsletter HTML contents.

Processing

The content is processed for links. They are saved for later statistics with name and URL in preparation for response tracking.

Output

The prepared newsletter to be sent.

Priority

Low

FR021: Order history

The client shall be able to view a list of recent orders. Drill down information should include order head along with order items. If there are many items, they should be divided into pages.

Input

The client clicks on a link.

Processing

The system lists all orders made in the shop, most recent first.

Output

A list of orders, with links to order details.

Priority

Low

General requirements

FR022: Front-end localization support

Description

The shop front-end shall on demand support viewing in different languages selectable by the visitor. Localization includes both general pages as well as product information pages. The client can choose which additional language to support, and then perform the translation of customize d pages, and all product related information.

Priority

Low

C.4 Non-functional requirements

Usability

NR001: Iterative usability testing

To ensure proper usability, the method of developing the user interface should involve iterative usability testing. The phases of designing and testing the user interface should be iterated at least three times, and finish when a fix point is reached, where the improvement in subjective satisfaction of the users is less than 10% between the iterations.

Priority

High

NR002: Usability testing methods

“Discount usability testing” should be performed to gain user feedback, as well as expert reviews using the “8 golden rules”.

Priority

High

NR003: On-line help

The MI shall provide short on-line help texts for parts of the interface that are not self-explanatory for the clients.

Priority

High

NR004: On-line start-up guide

A start-up guide shall be accessible on-line for the client. The guide should help the client to perform common tasks in a recommended order. The tasks should be organised in groups such as “Getting started”, “Page layout and contents” etc.

Priority

Normal

NR005: Browser compatibility

The interface should be usable with both Internet Explorer 6 and Mozilla Firefox 1.5. All tasks shall be possible to complete in both browsers and the design should look roughly the same.

Priority

Low

NR006: Screen resolution

The interface should be fully usable with a minimum screen resolution of 1024x768 pixels.

Priority

Normal

Performance

NR007: Simultaneous clients

At least three users from the same client should be able to work on MI2 simultaneously. They should each have their own login account.

Priority

Normal

NR008: Predictable response times

The response times of the interface should be about the same over the whole interface. If some page has a response time more than 2 seconds longer than the average page, there should be a message to the client before the page is loaded. Response time is the time from the point where the server receives a request, until the page starts loading.

Priority

Low

Compatibility with existing environment

NR009: Database environment

The database engine for the system is Microsoft SQL Server 2000 or later versions.

Priority

High

NR010: Web application environment

The system will be deployed on a Microsoft ASP.NET 2.0 platform, using IIS6.

Priority

High