



Department of Computer Science

Susanne Haase and David Götze

Development of an Application for Managing Speed Skating Events

C-level thesis

Date/Term: 2011-06-09
Supervisor: Katarina Asplund
Examiner: Donald F. Ross
Serial Number: C2011:04

Development of an Application for Managing Speed Skating Events

Susanne Haase and David Götze

This thesis is submitted in partial fulfillment of the requirements for the Masters degree in Computer Science. All material in this thesis which is not my own work has been identified and no material is included for which a degree has previously been conferred.

Susanne Haase and David Götze

Approved, 2011-06-09

Advisor: Katarina Asplund

Examiner: Donald F. Ross

Abstract

This thesis provides insight in the further development of an application for managing speed skating events, based on the previous developed application by a group of students in fall 2010. The application was implemented for the Karlstad Speed Skating Club, which before used a program from the 90's. After finishing the implementation, all basic functionality requested by the Karlstad Speed Skating Club can be used for managing speed skating events.

The first part of the thesis presents a short summary of our work, comparing what we wanted to implement and what we achieved. A short background on speed skating, the old system, developed by a group of students, and the project requirements are presented. An introduction to the used tools, languages, techniques and development processes is given along with details concerning the implementation of the requirements. Finally, the design of the new user interface is explained and the results are presented, containing an evaluation of the implementation, occurred problems and what could be added in the future.

Contents

1	Introduction	1
2	Background	3
2.1	Introduction to Speed Skating	3
2.2	The old System	6
2.2.1	Functionality of the old System	6
2.2.2	Architecture of the old system	10
2.3	Project Requirements	14
2.3.1	Basic Requirements	14
2.3.2	Additional Requirements	16
2.4	Summary	16
3	Product Development	17
3.1	Tools and Languages	17
3.1.1	Microsoft Visual Studio 2010	17
3.1.2	AnkhSVN	19
3.1.3	Microsoft C#.Net	19
3.1.4	Microsoft SQL Server Compact 3.5	19
3.1.5	Extensible Application Markup Language — XAML	20
3.2	Techniques	20

3.2.1	Three layer architecture	20
3.2.2	Windows Presentation Foundation — WPF	22
3.3	Development Processes	22
3.3.1	Waterfall model	22
3.3.2	Scrum	24
3.3.3	Test-Driven Development(TDD)	25
3.4	Implementation of the Requirements	26
3.4.1	Calculation of speed skating points	26
3.4.2	The starting list	27
3.4.3	The result list	30
3.4.4	Export to Microsoft Excel	31
3.4.5	Desktop application	33
3.4.6	Skaters, Events and Clubs	36
3.4.7	User manual	38
3.4.8	Edit and remove	38
3.4.9	Export and import of the database	39
3.4.10	Publishing the application	39
3.5	Summary	40
4	The new User Interface	41
4.1	Structure and Navigation	41
4.2	Layout	43
4.3	Page Description	43
4.3.1	Main Page	43
4.3.2	Events	44
4.3.3	Edit Event	45
4.3.4	Add Classes and Distances	46
4.3.5	Add Competitors	47

4.3.6	Starting List	48
4.3.7	Result List	50
4.3.8	Classes	51
4.3.9	Distances	52
4.3.10	Clubs	52
4.3.11	Skaters	53
4.4	Summary	55
5	Results and Evaluation	57
5.1	Results	57
5.2	Beta Testing	58
5.3	Problems	60
5.3.1	Relative connection string	60
5.3.2	Database connections	60
5.3.3	Adding new columns to the tables of the databases	61
5.3.4	Data bindings	62
5.3.5	Importing a new database	62
5.3.6	Publishing the application	62
5.4	Summary	63
6	Conclusion and Future Work	65
6.1	Experiences	66
6.2	Future work	66
	References	69

List of Figures

2.1	GUI of the old System	6
2.2	Reduced class diagram of the old system	10
2.3	Database Design of the old System	13
3.1	The designer in Microsoft Visual Studio 2010	18
3.2	General design of the three layer architecture	21
3.3	The traditional waterfall model [4]	23
3.4	An starting list for a race exported to Microsoft Excel	32
3.5	An result list for four distances of a class exported to Microsoft Excel	33
3.6	Design of the new database.	35
4.1	Structure of the new user interface.	42
4.2	The user interface of the main page.	43
4.3	The user interface of the events page.	44
4.4	The user interface for editing an event's information.	45
4.5	The user interface for adding classes and distances to an event.	46
4.6	The user interface for adding skaters to an event.	47
4.7	The starting list in the unlocked modus.	49
4.8	The starting list in the locked modus.	50
4.9	The user interface for the result list.	51
4.10	The user interface for adding classes.	52

4.11	The user interface for the club page.	53
4.12	The user interface of the skaters page.	54
5.1	User Interface Problems in Windows 7	59
5.2	Reduced class diagram of the new system	61

List of Tables

2.1	Requirements for a speed skating management program	5
2.2	Functionality provided by the old system	9
2.3	The basic requirements	15
2.4	The additional requirements	16
6.1	The tasks still left to do	67

Chapter 1

Introduction

In Karlstad, Sweden, each year the speed skating event Karlstad Nordiska takes place. Speed skating is a competitive form of ice skating, in which a lot of skaters compete against each other in different distances and classes, mostly according to their ages and gender. After the races the winners are decorated with medals.

Organizing speed skating events means a lot of work. First all classes and distances need to be assigned to an event. In the next step, the clubs can register their skaters to certain classes in the event. After all skaters are assigned to their classes the starting lists have to be created and printed. Finally, the event can take place and after each race the results are registered and sorted to create a result list, which is printed out.

Until now, the Karlstad Speed Skating Club has administrated the speed skating events with a very old program, started from a floppy disk on a laptop built in the beginning of the 90's. For understandable reasons the Karlstad Speed Skating Club wanted to have a new program, which would make the administration easier. In fall 2010 a group of students started to develop the new program and now we have finished the program as a part of our bachelor thesis.

The application has a high value for the customer, a member of the Karlstad Speed Skating Club, because we implemented all the basic functionality for managing speed

skating events, along with the requested special functionality the customer also requested. By including this functionality, our program supports the members of the Karlstad Speed Skating Club while creating and managing events and saves them a lot of time.

This thesis describes how we implemented the requirements. It also describes how speed skating races are organized, executed and explains exactly how the project is structured. The thesis also discusses tools, languages and techniques we used, with which development models we worked and the changes we did to the old system the students' group had developed. A whole chapter presents the new user interface, which we created using Windows Presentation Foundation (WPF). The rest of the thesis is structured as follows. Chapter 2 presents the background of the project, including an introduction to speedskating, a description of the old system as well as the project requirements. Chapter 3 covers the product development. Explanations of tools and languages as well as techniques we used are also given. The biggest part of the chapter describes the implementation of the requirements. In chapter 4 the new user interface is described in detail and chapter 5 gives a summary of the results we achieved and discusses the problems that occurred. Finally, Chapter 6 gives an evaluation of the project and describes possibilities for future work.

Chapter 2

Background

This chapter gives a brief introduction to speed skating, its rules and how the events are organized in Karlstad. A table of requirements is given, to show what is needed in an application for handling speed skating events. Moreover, the old system created by the group of students is described, including the general functionality and architecture of the system. Finally, the requirements for our project are listed after priority.

2.1 Introduction to Speed Skating

The Karlstad Speed Skating Club wanted us to create a program which would help them to organize their speed skating events. In the program, they needed a lot of functionality, which is necessary for such events.

Speed skating is a competitive form of ice skating. Competitors travel a certain distance on skates and race each other. The competitors are divided into speed skating classes according to their age and gender. In an event a class has four distances assigned, which can be different from each other. It can happen that a class has the same distance four times. Different classes can also have different distances assigned as well. For each distance in each class there is a starting list which can be created manually or by using different

sorting approaches. Those starting lists can, be do not need to be equal for all distances in the class. A starting list can be sorted in descending order by personal bests for the distance, by previously achieved results in the event, or completely randomly (see section 3.4.2). Initially, each skater needs to be part of all races belonging to the class she or he is registered to. After the race, the results are shown in a result list sorted after the classes and the skater's achieved times, starting with the best. The times the skaters achieved are converted into so called speed skating points (see section 3.4.1).

A race itself can be conducted with a maximum of four skaters starting at the same time. The skaters' times are measured either manually or automatically. In the manual way, three people with a stopwatch are measuring the time. The time which is used as the result is the mean time. In the automatic way the clock is triggered by a gun impulse and stopped using a laser at the end of the track, which results in a more accurate time. In Karlstad it is common to start with two skaters at a time and to stop the time using an automatic time tracking system.

The Karlstad Speed Skating Club needs to manage the organisation before an event. Therefore, the clubs have to hand in the names of the competitors, their personal bests together with the class they want to start in. The speed skating club saves this information and places the skaters in their classes. Additionally, a skater is able to start in more than one class. After all the competitors are registered, the race organisation can start. The starting lists are created and each skater is provided with a starting position and a lane (inner or outer) for each racing distance. Furthermore, the competitor receives a starting number which is independent of his class, race and club and lasts for the whole event. Before the race a referee needs to verify all starting lists.

A system which can handle an event like this needs to fulfill special requirements. Table 2.1 gives an overview of the most important requirements. These are the requirements which are necessary for the customer. In addition, a good and intuitive user interface is very important.

Requirement	Details
1. Events	<ul style="list-style-type: none"> - Create events for a specific date and place - Add classes to the event - Add four distances (those could also be the same one) to each class in the event - Add competitors to classes in the event
2. Starting list	<ul style="list-style-type: none"> - Create a starting list for a race in an event, containing all registered competitors - Have the possibility to sort the starting list by personal best, previously achieved results in same event, completely randomly - Have the possibility to edit the starting list by moving the skaters up and down - Have the possibility to enter results in the starting list including statuses: pending, completed, personal best, personal best/fell, fell, did not start, did not finish, disqualified - Convert result into speed skating points automatically - Automatical check, if result is a personal best of the skater and see it appear as status - Automatical check, that invalid results (did not start, did not finish, disqualified, pending) always have the time 00:00:00 as result
3. Result list	<ul style="list-style-type: none"> - Create a result list for a race, a class in an event or the whole event - See all results sorted according to the earned speed skating points and status
4. Skaters	<ul style="list-style-type: none"> - See all skaters in the system - Add new skaters - Store details: name, gender, personal number, club - Add a personal best which is not connected to an event - Set a skater to active or passive
5. Clubs	<ul style="list-style-type: none"> - See all clubs in the system - Add new clubs - Store details: name, country, website, contact name, contact mail - See all skaters of a club
6. Edit and remove	<ul style="list-style-type: none"> - Provide edit and remove functionality for classes, distances, events, clubs, members, results and personal bests

Table 2.1: Requirements for a speed skating management program

2.2 The old System

This section gives information about the old system a former group of students developed within their lab-work project. It describes what they achieved and what was still missing.

2.2.1 Functionality of the old System

To provide the functionality described in section 2.1, the former group developed a web based system. This system had a lot of basic functionality, but did not provide a high value for the customer. It was implemented using C# (see section 3.1.3) and the user interface was web based (see Figure 2.1).

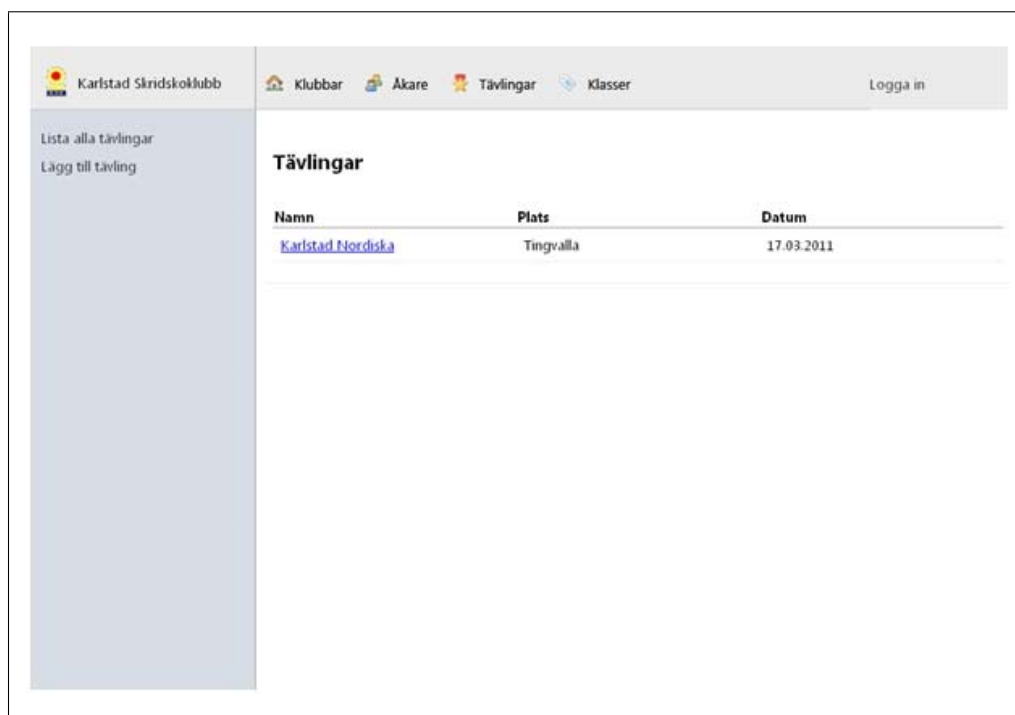


Figure 2.1: GUI of the old System

Four menus covered all the functionality for clubs, skaters, events and classes.

In the 'club menu', the user could see the list of all clubs, as well as create a new one. The user could also select a club to see all relevant information about the club and the

skaters assigned to it. Besides editing the information of the club it was possible to add new personal bests for a specific distance to club members. The user could create a new skater within the club menu and add him or her to a previously selected club. In this way, the club menu covered two parts of functionality in one. It managed both the clubs and the skaters that were part of the club.

The 'skater menu' offered a list of all skaters registered as well as possibilities to edit the skater's information. Adding skaters was only possible within the 'club menu', because every skater needed to be a member of a club

The 'event menu' gave the opportunity to see all current and past events as well as to create a new one. Furthermore, the user could select an event and add skating classes and distances to this event. In this menu there were still parts of functionality missing: the user could not add the same distance more than one time to a class. Subsequently, the user could select a speed skating club and a speed skating class and register the club's skaters in the selected class in the event. However, it was not possible to add skaters without the knowledge of their belonging club. The functionality for displaying the result and the starting list was started but did only provide a list of skaters registered to the class. There was no sorting by ranking, by starting position or completely randomly. The skaters always appeared in the order they were added to the class. The entering of results was also started but did not work at all, because the starting list was not finished.

Within the 'class menu' all existing skating classes were listed and the user was able to create a new one.

There was no menu item for the distances and it was not possible to add new distances within the interface. This meant that the user could not create an event by using the user interface.

To summarize, the old system was not usable for the customer, because a lot of functionality was missing. Especially the most important functionality, the proper creation of a starting list, the entering of the skater's results, the creation of a result list and the ability

to print out or export the lists were not implemented. In addition, the user interface was not intuitive. To give an example: Adding new skaters was not possible in the 'skaters menu', only in the 'club menu'. Nevertheless, the system was a good basis, because a lot of smaller parts of functionality were implemented. However, more complex parts were missing, so it was not possible to manage events.

To visualize the finished and still missing parts Table 2.2 again lists the requirements. The finished requirements are marked, while the ones still missing are crossed out.

Requirement	Details
1. Events	<ul style="list-style-type: none"> - Create events for a specific date and place - Add classes to the event - Add four distances (those could also be the same one) to each class in the event - Add competitors to classes in the event
2. Starting list	<ul style="list-style-type: none"> - Create a starting list for a race in an event, containing all registered competitors sorted after starting position - Have the possibility to sort the starting list by personal best, previously achieved results in same event, completely randomly - Have the possibility to edit the starting list by moving the skaters up and down - Have the possibility to enter results in the starting list including statuses: pending, completed, personal best, personal best/fell, fell, did not start, did not finish, disqualified - Convert result into speed skating points automatically - Automatical check, if result is a personal best of the skater and see it appear as status - Automatical check, that invalid results (did not start, did not finish, disqualified, pending) always have the time 00:00:00 as result
3. Result list	<ul style="list-style-type: none"> - Create a result list for a race, a class in an event or the whole event - See all results sorted according to the earned speed skating points and status
4. Skaters	<ul style="list-style-type: none"> - See all skaters in the system - Add new skaters - Store details: name, gender, personal number, club - Add a personal best which is not connected to an event - Set a skater to active or passive
5. Clubs	<ul style="list-style-type: none"> - See all clubs in the system - Add new clubs - Store details: name, country, website, contact name, contact mail - See all skaters of a club
6. Edit and remove	<ul style="list-style-type: none"> - Provide edit and remove functionality for classes, distances, events, clubs, members, results and personal bests

Table 2.2: Functionality provided by the old system

2.2.2 Architecture of the old system

The old system was a server program based on the three layer architecture (see section 3.2.1). The architecture splits up the program into the 'Presentation Layer', which includes the user interface and the input validation, the 'Domain Logic Layer', which includes the whole business functionality of the program (therefore it is often called 'Business Layer') and the 'Data Access Layer' which maps the data to a data source. This is done using the mapper classes in this layer. Figure 2.2 shows a reduced class diagram of the old system to illustrate how the three layer architecture was implemented.

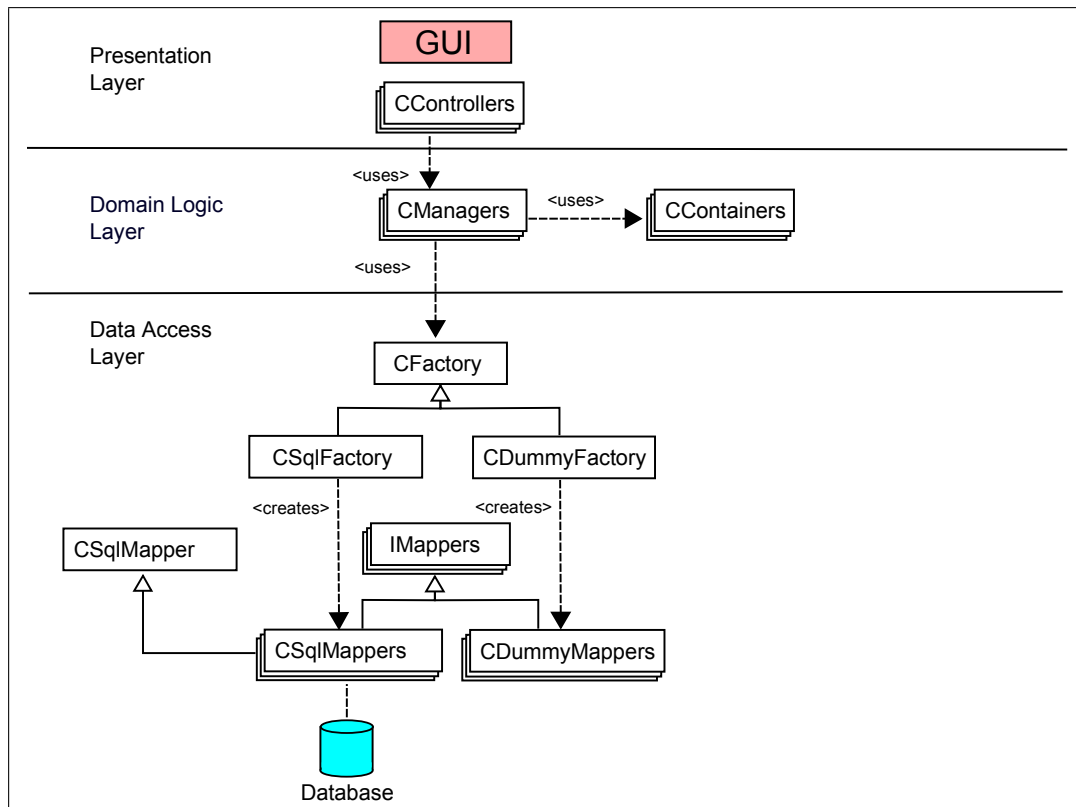


Figure 2.2: Reduced class diagram of the old system

The old system also consisted of a test project with several test classes for testing the entire 'Domain Logic Layer' as well as the 'Data Access Layer'. Below follows a description of the three layers.

The Presentation Layer

The 'Presentation Layer' of the old system was broken down into 'Views', which are small parts of the user interface and 'Controllers', which contain the input validation for one or more 'Views'. In the old system the 'Views' were split into small groups. Each group covered one small part of the program's functionality, for example add and edit as well as showing the information about a club. Each of those small groups of 'Views' had one 'Controller' assigned, which validated the input and then forwarded the input data to its belonging manager class in the 'Domain Logic Layer'.

The Domain Logic Layer

The 'Domain Logic Layer' consisted of manager and container classes. A manager class contained only a small amount of overall functionality, which mostly covered one point of interest, for example a club or a speed skating class.

For saving, updating or retrieving data from the data source the manager class called a method in its belonging mapper class in the 'Data Access Layer'. The retrieved data was then stored in an instance of a container class, which was representing for example a club.

The Data Access Layer

The 'Data Access Layer' handled all the communication between the program and the data source. The program had two different data sources. One was a MySQL database, which was stored on a web server. The other data source was used for testing purposes and therefore consisted of lists, which only held the data while the tests were executed.

Each data source had several classes, which just covered a small amount of the overall functionality. It was often connected to one table in the database or list in the test data source. One mapper class inherited from its belonging interface and covered only the functionality of one data source. There were basically two classes with nearly the same functionality but for different data sources.

The purpose of using two data sources was to speed up testing. Therefore, both data sources were fully tested. For testing the managers the testing data source was used.

Design of the Database

The old system used a MySQL database, which was designed as shown in Figure 2.3.

The entries in the tables for users, classes, events, distances, starting lists and clubs were independent from other tables, while the entries in the tables for member, competitors and races were connected to nearly all the tables. As shown in the Figure 2.3, the members, representing a skater, had a foreign key pointing to a club. This was done due to the fact, that no skater can compete in a speed skating event if she or he is not part of a club.

A race is the connecting element between an event, a class, a distance and a starting list. Therefore, the entry for a race has foreign keys pointing to the tables for events, distances, classes and starting lists.

An entry in the competitor table is the connecting element between a member, a race and the starting list. The entry also contains the information about the skater's result in the race. Therefore, a row in the competitor table has foreign keys pointing to the database rows of its belonging member, race and starting list. The foreign key for the starting list was included here, because the old system should have included the functionality to move skaters between starting lists. Therefore, it was possible for a competitor to appear in a starting-list not belonging to the race she or he is competing in.

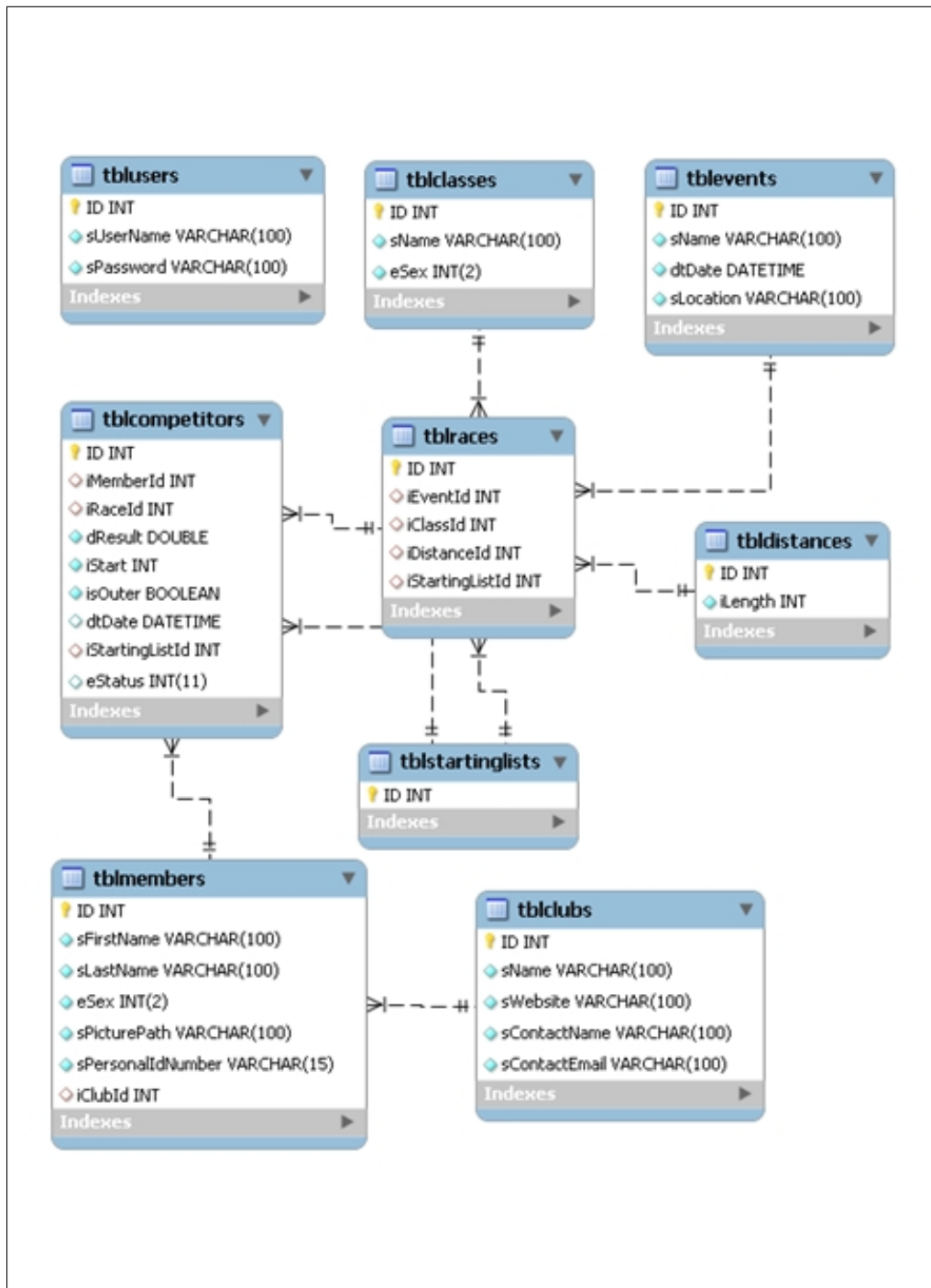


Figure 2.3: Database Design of the old System

2.3 Project Requirements

By negotiating with the customer, a member of the Karlstad Speed Skating Club, we decided to focus on the prioritized requirements listed below (see Table 2.3). Due to frequent meetings with the customer the project requirements and their priority changed from time to time. If the basic requirements had been finished before the end of the project, we should have worked on the additional requirements. Besides implementing the basic functionality, the system should also be transformed from an online server application to a desktop application, which could be executed on a computer without internet connection. This transformation needed to be done because Karlstad's speed skating arena does not have an internet connection via cable and the customer did not want to rely on an internet connection using a UMTS stick. This change required a complete revision of the user interface, which until that point was an interface in a web browser, as well as a new database.

2.3.1 Basic Requirements

Table 2.3 contains the basic requirements sorted after priority. The requirements marked with a 'X' are the ones from Table 2.2 which were crossed out and were not implemented in the old system. The tasks marked with '-' are additional tasks that the customer wanted as basic requirements even though they are not necessary in an application for managing speed skating events.

Requirement	Description
1. Speed skating points	X Calculate speed skating points as described in Appendix A1
2. Starting list	X Create a starting list sorted randomly, by personal best or by previous achieved results in same event
	X Editing the starting list by moving a skater up and down
	- Locking and unlocking the starting list
	- Locking the starting list should disable sorting and enable entering of results and export functionality
	- Unlocking the starting list should enable sorting and disable entering of results and export functionality
	X Enter a skater's result in the starting list, including status and automatically check if they are a new personal bests and set invalid results to time 00:00:00
	- Add the status Personal best/Fell
3. Result list	X Entered results should appear in the result list sorted after achieved time
	X Create result list for one race, one class or the whole event
4. Export functionality	- Export a starting list to Microsoft Excel
	- Export a result list to Microsoft Excel
5. Desktop Application	- Adapt the old system so that it works without internet connection
	- Create a new database and a new database connection
	- Create a new user interface
6. Skaters	- Add a personal best which is not connected to an event
	X Set a skater active or passive
7. Event	- Set an event current or previous
	X Add four distances (could also be the same one) to each class in event.
	- Show the number of skaters participating in one class of an event
8. Club	- Add country to club information
9. User Manual	- Continuously write a user manual covering all the system's functionality
10. Edit and remove	X Provide edit functionality for events, clubs and skater
	X Remove functionality for classes, distances, events, clubs, skater, competitors and personal bests not connected to an event
11. Reusability	- Document the code so that another group can continue
12. Database	- Provide export and import of database
13. Installing the application	- The final product should be installed using a setup file

Table 2.3: The basic requirements

2.3.2 Additional Requirements

Additional requirement	Description
1. Starting list	- Provide withdraw functionality
2. Online Application	- Reduce the functionality of the online application to managing clubs and skaters. Provide functionality to register a skater to a class in an event.
	- Each club should get a user name and password to manage the skaters this club, its information and registration the skaters for new event on his own.
3. Starting Fee	- Provide functionality to manage the starting fees for an event
4. Relay	- Include relay functionalities
5. Provide informationl	- Add phone number, country, city, club logo to a club
	- Add website, Facebook, Twitter, Email, phone number to a skater
	- Add photo of the arena to the event

Table 2.4: The additional requirements

2.4 Summary

This chapter all the necessary background information was given, including a short introduction to speed skating. The functionality as well as the architecture of the old system were described and it was shown why this system was not usable for the customer. Finally our project requirements and the functionality we were asked to implement were listed.

Chapter 3

Product Development

The following chapter describes our approach to fulfill the project requirements (see section 2.3). It first describes the used tools and languages, followed by the development processes and techniques we used during the project. Furthermore, it gives a detailed overview of the changes done to the old system's 'Data Access Layer' and 'Domain Logic Layer'. The new user interface is described in chapter 4.

3.1 Tools and Languages

This section covers a description of Microsoft Visual Studio 2010, which was chosen as our development environment, the Visual Studio subversion plugin AnkhSVN, the programming language C#.NET, SQL Server Compact 3.5, as well as the description language XAML.

3.1.1 Microsoft Visual Studio 2010

To develop our project we needed a development environment that was capable of handling the old system, which was written in C#. We decided to use Visual Studio 2010 Professional. This tool provides an easy and efficient C#-development, and the user friendly

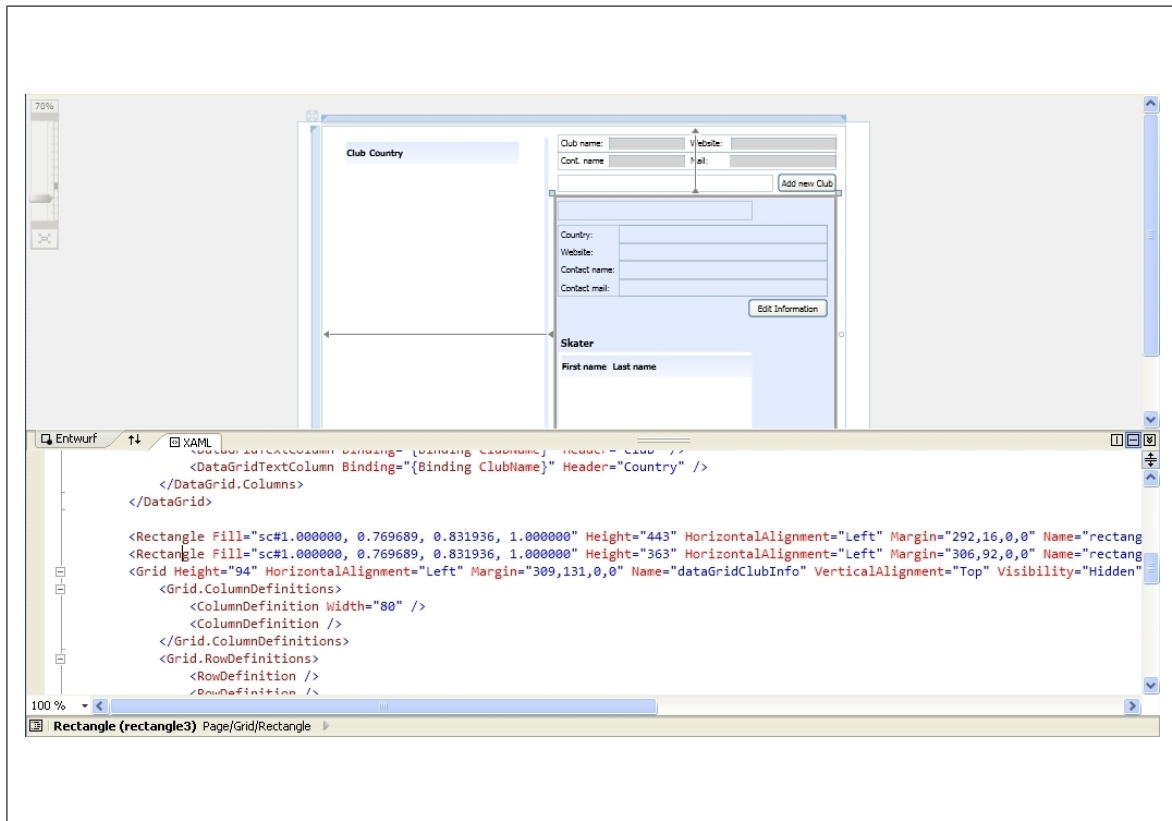


Figure 3.1: The designer in Microsoft Visual Studio 2010

user interface is also easy to handle. There is, for example, a designer with whom creating Windows Presentation Foundation (see section 3.2.2) Windows and Pages is made easy and fast. On the upper half on the designer window, a preview screen is visible and below this screen the XAML-code (see chapter 3.1.7) of the Window or Page is shown (see Figure 3.1). The time needed to design the elements, like buttons or tables, programmatically is reduced due to the simple drag-and-drop functionality for adding and editing parts of the user interface. Useful also are the auto-complete function, the included debugger and the testing environment. Because we worked as a team and therefore on different computers, subversion control was necessary. We used a subversion tool (see section 3.1.2), which made the synchronization easier. Visual Studio 2010 also provides a tool for creating databases in SQL for Microsoft SQL Server Compact 3.5 (see section 3.1.4).

3.1.2 AnkhSVN

AnkhSVN is a subversion support tool for Microsoft Visual Studio. Subversion can be used to store a project solution on a central server. In our case it was a server at Karlstad university. Each user who is working with this system can 'check out' the solution and implement functions. When she or he is done, the solution can be committed to the server again. It is also possible to commit only one class or a project map of the project. The only problem that appears from time to time is when two or more users work on the same file at the same time and commit the changes done to it. Then a conflict can appear, which has to be sorted out manually. However, AnkhSVN is supporting the user with that. It allows the user to do the subversion management directly from inside Visual Studio. Additional version control is also possible. If something has gone wrong an old version can be restored easily.

3.1.3 Microsoft C#.Net

In order to continue with the project we decided to use C#, in which the old system was written. C# is developed by the Microsoft Company within the .NET initiative. It is "encompassing imperative, declarative, functional, generic, object-oriented (class-based), and component-oriented programming"[6] language.

3.1.4 Microsoft SQL Server Compact 3.5

"SQL Server Compact 3.5 SP2 is a free, easy-to-use embedded database engine that lets developers build robust Windows Desktop and mobile applications that run on all Windows platforms including Windows XP, Vista, Pocket PC, and Smartphone."[3]

The above quote states most of the benefits of SQL Server Compact 3.5, a relational database from Microsoft. On the other hand, this piece of software has a very limited

SQL-syntax and database size, which is limited to 4 GB. However, it is very simple to use as a database engine.

3.1.5 Extensible Application Markup Language — XAML

Chriss Sells and Ian Griffith described the markup language like this: "XAML — the eXtensible Application Markup Language — is an XML-based language for creating trees of .NET objects. XAML provides a convenient way of constructing WPF user interfaces." [5]

XAML is a new developed description language for designing user interfaces especially for WPF (see section 3.2.2). XAML describes hierarchies of the user interface with the help of XML declarative. To each XAML - class, which only defines the user interface, belongs a C# class, which contains the input validation and logic. The C# class gets access to the components of the user interface by using the identifiers for the components defined in the XAML class.

3.2 Techniques

The following section covers the techniques that were used while working on the system.

3.2.1 Three layer architecture

The three layer architecture is an architectural pattern that separates the software in three layers called the 'Presentation Layer', the 'Domain Logic Layer' (also called 'Business Layer') and the 'Data Access Layer', see Figure 3.2. Usually the layers are connected using interfaces. The layering is done to support an easy exchange of for example the 'Presentation Layer' or the 'Data Access Layer', without a need to change something in the 'Domain Logic Layer'. What we have changed within all the layers in comparison to the old system is explained in detail in section 3.4.

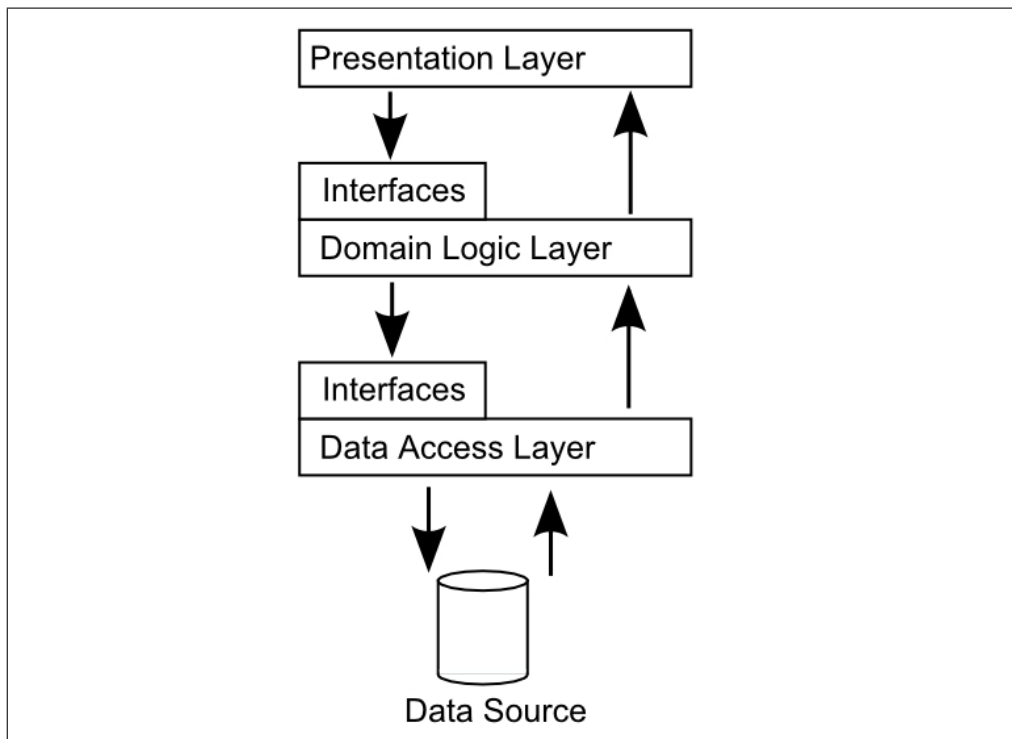


Figure 3.2: General design of the three layer architecture

By separating the core of the software from the user interface and the data source, the program becomes more flexible and the code can be reused in other pieces of software or in another version of the same software using a different data source or user interface. Below follows a description of the three layers.

The 'Presentation Layer' is the topmost layer in the architecture. It contains the user interface and the input validation. For retrieving, manipulation or deleting data the 'Presentation Layer' uses methods provided in the 'Domain Logic Layer'.

The 'Domain Logic Layer' is the mid-layer and contains all the logic connecting the different tables of the database, retrieves their information by accessing the mappers, combines them and puts them into container objects. These container objects can then be forwarded to the user interface.

The 'Data Access Layer' is the lower layer in the three layer architecture. It provides functionality for retrieving, removing and manipulating data in one or more tables of the database.

3.2.2 Windows Presentation Foundation — WPF

WPF [5] is a graphical framework which is contained in Microsoft's .Net framework. WPF applications can be desktop as well as web applications. In WPF exists a separation between user interface and the logic behind¹, which is supported by the possibility to use XAML (see section 3.1.5) for creating a user interface hierarchy.

A nice feature is that Windows-Forms² elements can be used within WPF-applications. We used this frequently. Using WPF, objects can directly be bound to view components like for example datagrids. This binding is done by defining the properties belonging to, for example, a data grid in the XAML file. Filling the data grid simply consists of handing over a list of objects. This was a welcome advantage for us.

3.3 Development Processes

The following sections cover the development processes used. The first part gives a brief introduction to the waterfall model and Scrum, which is the development process we used. Afterwards, there is a part about Test Driven Development, which helped us to produce code with less errors and bugs.

3.3.1 Waterfall model

The development of a program is traditionally done using the so called waterfall model (see Figure 3.3). This model was developed by Dr. Winston W. Royce in 1970. In this model,

¹Read about the separation within section 3.1.5.

²Windows-Forms is a programming interface for creating graphical user interfaces. It is part of the Microsoft .NET framework.

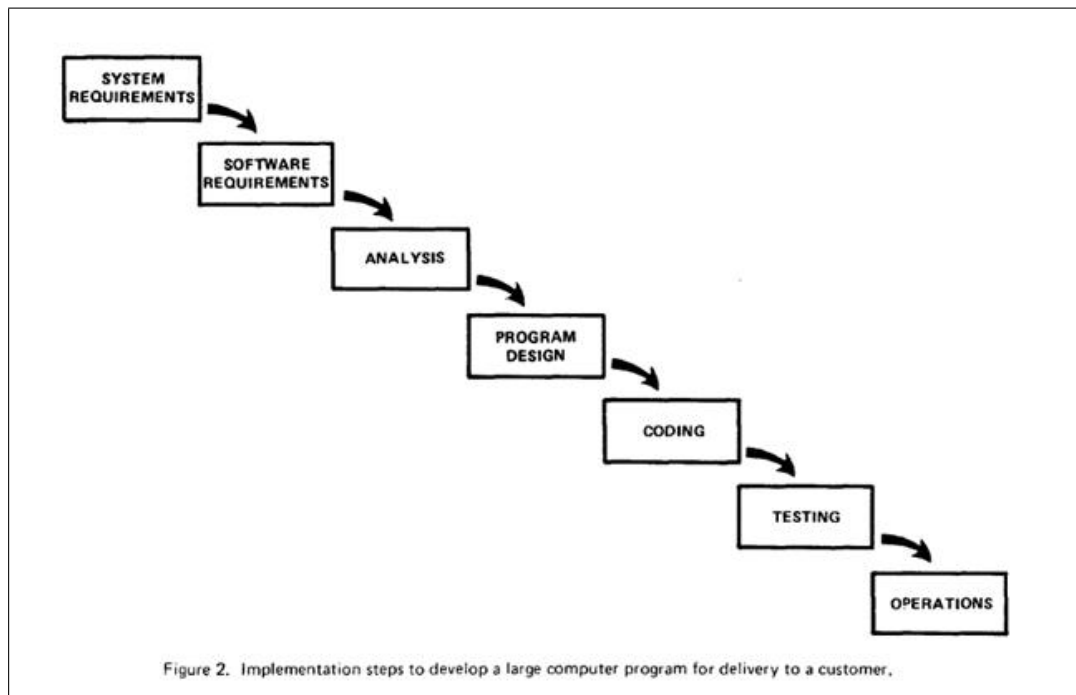


Figure 3.3: The traditional waterfall model [4]

the program requirements are first analysed, then the program is designed and coded and afterwards it is tested and maintained. Going steps backward in the model is not possible. Although this model has been widely used in software development this kind of developing very often results in big problems, because it could happen that the delivered product does not match the demands of the customer or that the costs far exceeds those expected. [4]. The waterfall model was a first approach for managing the development process for large software systems, nevertheless the method "[...] is risky and invites failure." [4] Already in 1989 John C. Kunz wrote the following about the disadvantages of the waterfall model:

"Software projects are often delivered later and are more expensive than planned, and they often lack intended functionality and include bugs. One source of these software development difficulties is that requirements are difficult to specify precisely, and they often change as users see new possibilities and developers identify new ways to represent and analyze problems. Thus, while the waterfall

method is simple to describe and easy to use in principle, it alone has not been sufficient to provide the basis for effective software development.” [2]

A newer and more effective working method is Scrum, which is explained in the next section.

3.3.2 Scrum

Scrum is a development model that differs from the traditional model. Instead of doing the whole development from top to bottom, the system is developed in small parts, where each part is developed in its entirety, one at a time. The team members work iteratively and add new functionality to the system in each iteration.

The model entails working in small programming cycles, where the whole programming work is divided into small sprints. In each sprint, the team develops a small part of the system, which should be well tested and ready to be deployed at the end of the sprint. A sprint should always have a fixed length, depending on the time box the team has chosen in the beginning of the project. Each sprint starts with a big planning meeting and is therefore well organised by the team. [1]

The team itself is cross-functional, which means that not a single person is responsible for example the database or for the design, but the whole team is working on every part. The team is supposed to have daily stand up meetings at which all team members are informed about what is left to do and what should be done on this particular day. The stand up meetings are also the platform for discussing problems.

In Scrum it is necessary to have a good and frequent contact with the customer. After each sprint the customer can see the result and should tell his opinion about it. If there are some changes to be done, they can be included in the upcoming sprint. This method helps to develop software the customer really wants. Another advantage is that the customer has always the possibility to stop the work if the costs are exceeding the budget too much. She or he will still get a program, with a part of the requested functionality. With the waterfall

model this is not possible, because the program is done stepwise from the beginning to the end. Therefore, the program is not executable until everything is ready and tested, which is the last step in the development of the waterfall model.

Because we were only two persons, we decided to use Scrum in our own way. We have chosen to work in two week sprints and meet the customer afterwards. The meetings turned out to be very helpful, because we had the chance to pinpoint problems and rework parts of the application. We did not do stand up meetings, because we were always sitting side by side in one room and knew what the other one was doing. After each sprint we had a small working program with a part of the functionality. After the meeting with the customer we decided, what to do within the next sprint.

A big part of Scrum is Test-Driven Development, which ensures the customer that the software is working properly.

3.3.3 Test-Driven Development(TDD)

Enrik Kniberg wrote a short summary of TDD which explains it very well:

”Test-driven development means that you write an automated test, then you write just enough code to make that one test pass, then you refactor the code primarily to improve readability and remove duplication. Rinse and repeat.” [1]

TDD helps to produce code which is well tested. Before writing code, a test is written. It is only as much code written as needed to pass the test. This method of working prevents the developer from writing unnecessary and untested code.

We tried to work with TDD as often as possible. Nearly all our functions are therefore tested. It is a very useful practice even though it can be annoying from time to time. However, if there are some changes done to the system, which affect more than just one part, it is very helpful. The programmer can see exactly which part of the program causes the error, just by looking at the failing tests. This means that solving them afterwards is

made easier.

3.4 Implementation of the Requirements

The following sections cover the implementation of the requirements listed in Table 2.3. The whole work described in this section is based on the architecture and functionality provided by the old system. We continued to use the three layer architecture (see section 3.2.1) and did intense testing using TDD (see section 3.3.3). For the testing we reused the test project and the included tests from the old system and extended it.

3.4.1 Calculation of speed skating points

The most important part to start with was the calculation of speed skating points from the results a skater has achieved in one or several races. The calculation needed to be done first, because the implementation for sorting the starting list is based on it.

The calculation of speed skating points is based on the 500-meter times of a skater. The achieved time of a skater is divided by the result of the division of the distances length and 500 meter. This means that the speed skating points for a race represents the average amount of time a skater needs to cover a distance of 500m. The calculation is done using the following equation:

$$Points = \frac{Time}{\frac{Length}{500}} \quad (3.1)$$

After calculating the speed skating points they are saved in a new created column in the database table of the competitors. This caused a problem described in section 5.3.3. For further information on the calculation of speed skating points see Appendix A.1.

3.4.2 The starting list

The next important part to implement was the starting list. As explained in section 2.1. a starting list represents the starting order of skaters for one race in a class. To support a realistic representation of the starting list and competitors of a race, new columns needed to be added to the database table concerning the competitors, representing the starting number and the starting position. This again caused the problem described in section 5.3.3.

The starting list displays the skaters in pairs. If the number of skaters is odd the first pair only consists of one skater. The starting order could be generated following different sorting algorithms and, by request of the customer, was only used to create the starting list for one race in a class. This provides the possibility to have different starting lists in one class sorted using different algorithms. The algorithms we implemented are described below.

Sorting the starting list randomly

The first possibility to create a starting list is to pair up all skaters completely randomly. The following algorithm is used:

All skaters associated with a certain starting list are retrieved and randomly sorted. Thereafter, the number of skaters in the starting list is checked to see if it is odd or even. For an even number of skaters all skaters are put into pairs. For an odd number of skaters the first pair to start consists only of one skater and the rest of the skaters are put into pairs of two skaters.

Sorting the starting list by personal best

The second possibility to create a starting list is by sorting the skaters according to their personal best for the distance. All skaters are sorted in ascending order by their personal best and put into a list. Skaters without a personal best are added to the end of the list.

To reduce the chance of two skaters, having a nearly equal personal best, skating against each other in each event, the complete list of skaters is divided into boxes of a certain size. The box size is predefined with a maximum of four skaters. The skaters are sorted randomly within those boxes. The box containing the skaters with the best personal bests is at the beginning of the list. To make a speed skating event more exciting the whole list is reversed, so that the skaters with the best personal bests are at the end of the list. The skaters are then paired up according to the total number of skaters in the starting list.

Sorting the starting list by previous achieved results

The third possibility to create a starting list is to sort the skaters by their previously achieved results in the selected class of the event. In this way, the skaters can be sorted according to their actual performance. To accomplish this, all results of a skater in the selected class are added up. This is not done by just adding up the achieved times, but by adding up the speed skating points a skater earned in each race. Then the skaters are sorted ascending according to the sum of the previously achieved speed skating points. All skaters without previous results in this class are added to the end of the list. The list is then reversed and the skaters are paired up. If the total number of skaters is odd, the first pair only consists of one skater.

Edit the starting list by moving a skater up and down

Each starting list needs to be confirmed by a referee and therefore must meet certain criteria. Because the referee decides which criteria are important for a certain event and how the starting list needs to be edited to meet the criteria, a manual editing of the starting list was needed. This editing functionality basically consists of the ability to move a skater up or down in the starting list and was achieved by switching the starting and track position with the skater above or below.

Locking and unlocking the starting list

After the starting list is confirmed by the referee it can be locked, which means set to read-only. Locking the starting list disables all functionality for sorting the starting list, moving a skater within it and for adding new skaters to the class containing the race the starting list belongs to. The starting list can also be unlocked, if a user accidentally locks it. After the starting list is set to read-only, results can be entered in the starting list. When the first results are entered the starting list cannot be unlocked anymore. To unlock it again, all results need to be reset to the time 00:00:00 with the status 'Pending' (see below).

Enter a skater's result in the starting list, including status

To support a fast and easy entering of results, these are entered into the starting list. This approach avoids the search of skaters in a list, because the starting list shows the pairs in starting-order. Usually the results are added along with a status, which for example displays if the skater fell during the race. All in all there are eight statuses which can be associated with a skater's result:

- 'Pending' means that the skater has not started yet, but is still part of the starting list. A result that is entered using the status 'Pending' is invalid and reset to 00:00:00, because no skater can have a result without having started and finished a race.
- 'Personal Best' means that a skater achieved a personal best. This status cannot be set manually, but is automatically set by the system by comparing a result with the status 'Completed' with the current personal best times.
- 'Completed' means that a skater has successfully finished a race without any disturbance. The result is valid and therefore is compared to the current personal best. If the result is better than the current personal bests for the distance, the status is changed to 'Personal Best'.

- 'Personal Best/Fell' states that a skater achieved a personal best, even though he fell during the race. This status also cannot be set manually, but is added automatically if the result has the status 'Fell' but is better than the skater's current personal best. This status is new and was not available in the old system.
- 'Fell' states that a skater fell during a race but was able to finish the race. This result is valid and therefore is also compared to the skater's current personal bests and the status is set to 'Personal Best/Fell' if the result is better than the current one.
- 'Did not start' states that a skater withdrew from the race. This status makes the associated result invalid which is automatically set to 00:00:00 again.
- 'Did not finish' states that a skater started a race but was not able to finish. Because the skater did not cross the finish line there cannot be a valid result, therefore the result is set to 00:00:00.
- 'Disqualified' states that a skater was punished for violating one or more rules before or during a race. This result is again invalid and therefore reset to 00:00:00.

3.4.3 The result list

The result list shows the ranking of the skaters for one race. All skaters are sorted ascending according to their speed skating points. At the end of the list are the skaters with invalid results. The result list of the old system just displayed the skaters registered in the race without any sorting. The new user interface shows the result list for one race. To get results for one class or the whole event, the user needs to export the result list to Microsoft Excel.

3.4.4 Export to Microsoft Excel

At a speed skating event the starting and result lists are often printed out and put on a board where everybody can see them. To support this, the starting and result lists needed to be exported to a printable file. Because the customer uses Microsoft Excel frequently and wants to edit the visibility of the skater's information in the list, we decided to export the list to Microsoft Excel. The implementation of the export functionality uses a framework which is provided within Microsoft Office 2007. Therefore, the user needs to have Microsoft Office 2007 installed in order to use the export functionality. The export is done by creating an Excel file and filling it with data of a list. After creating the headline and adding the column headers the skater's information are put into the rows by iteration over the list of data. Finally, the file is saved to a previously selected location.

Export of the starting list

Starting lists can only be exported one at a time. The algorithm for exporting the starting list consists of an iteration over the competitors in the race. The exported starting list is sorted by the starting position and begins with the skater who starts first. Depending on, if the starting list is published before or after the race, the user can decide if it should be exported with the results and statuses of the skaters or with the old personal bests. Figure 3.4 shows a exported starting list for a race including the results and statuses of the skaters opened in Microsoft Excel.

Export of the result lists

For exporting a result list several options are available. The first option provides the possibility to export one or more races of a class. Therefore, the user needs to select which distances the result list should contain. Depending on this selection the created file contains a combined result list of all the selected distances in the class. The combined result list is sorted ascending by the sum of the achieved speed skating points. The skaters

	A	B	C	D	E	F	G
1	Karlstad Nordiska Tingvalla, Karlstad 2010-11-27						
2	<i>Starting List for Pojkar 11-12 - Distance 1</i>						
3	Pair	Lane	No.	Name	Country	Club	Personal Best
4	1	0	5	Mathias Solberg	NOR	Kongsvinger Skøyteklubb	00:59:74
5	2	1	2	Max Sjöstedt-Eriksson	SWE	IF Thor	01:00:07
6	2	0	7	David Karlingsjö	SWE	SK Trollhättan	00:53:92
7	3	1	3	Isak Höiby	NOR	Kongsvinger Skøyteklubb	00:53:92
8	3	0	4	Per Henry Röttum	NOR	Kongsvinger Skøyteklubb	01:19:79
9	4	1	6	Kevin Jacobsson	SWE	Motala AIF	01:10:45
10	4	0	1	Erik Wetterdal	SWE	IF Thor	01:25:78
11							

Figure 3.4: An starting list for a race exported to Microsoft Excel

with one or more invalid results are again added to the end of the list.

Besides choosing the distance, the user can also select if she or he wants to create a combined result list including the selected distances for the currently selected class or for all classes in the event. The exported list then covers combined result lists for each class, which are displayed one below the other. The user has, for example, the possibility to create a combined result list for a class covering the first three distances. The list will then be sorted after the combined results of all three distances of the class, starting with the skater with the best results in all races and who has only valid results. The algorithm for exporting the result list supporting these two options is similar to the one for exporting the starting list. The only difference is that the list is nested and therefore, the iterations are interleaved. Figure 3.5 shows an exported result list for all four distances of a class opened in Microsoft Excel.

	A	B	C	D	E	F	G	H	I
1	Result List								
2	Karlstad Nordiska Tingvalla, Karlstad 2010-11-27								
3	Place	Name	Club	Country	Dist 1	Dist 2	Dist 3	Dist 4	Points
4									
5									
6		Pojkar 11-12			500m	300m	500m	1000m	
7	1	Max Sjöstedt-Eriksson	IF Thor	SWE	01:00:42	00:36:00(PB)	01:00:56	01:00:07(PB)	211,015
8	2	David Karlingsjö	SK Trollhättan	SWE	00:55:23	00:45:01(PB)	01:00:01	00:53:92(PB)	217,217
9	3	Mathias Solberg	Kongsvinger Skøyteklubb	NOR	01:02:12	00:51:00(PBF)	00:59:76	00:59:74(PB)	236,75
10	4	Per Henry Röttum	Kongsvinger Skøyteklubb	NOR	01:31:04(F)	01:00:00(PBF)	01:15:71(PB)	01:19:79(PB)	306,645
11	5	Erik Wetterdal	IF Thor	SWE	00:00:00	00:29:89(PB)	01:02:34(PB)	00:00:00(DQ)	112,157
12	6	Isak Höiby	Kongsvinger Skøyteklubb	NOR	00:00:00(DQ)	00:46:00(PB)	00:55:61	00:53:92(PB)	159,237
13	7	Kevin Jacobsson	Motala AIF	SWE	00:00:00	00:52:12(PB)	00:59:98(PB)	02:50:33(PB)	232,012
14									

Figure 3.5: An result list for four distances of a class exported to Microsoft Excel

3.4.5 Desktop application

The new database and database connection

The database of the old system was a MySQL database, which was deployed on a webserver. Because the system should be transformed into an application, which can be used without internet, a new local database was needed. The old system was designed using the three layer architecture (see section 3.2.1). The architecture made it easier to change the data access and storage because only one part of the system, the 'Data Access Layer', needed to be changed. Due to the similarity between MySQL and SQL Server the existing code for the connection to the MySQL database could be reused. To support future work on the system the MySQL database connection was left within the project and the new SQL database connection was added using a modification of the old database connection. Due to the limited SQL syntax of Microsoft SQL Server Compact 3.5 (see section 3.1.4) some SQL queries had to be changed and some methods needed to be added.

The tests for the database connection were copied and modified to fit the SQL database. The SQL database was created using a tool within Microsoft Visual Studio 2010 (see section 3.1.1). To test the whole project all database connections had to be tested. Therefore

each method in the 'Data Access Layer' had to be implemented three times, for the SQL database connection, for the MySQL database connection and for the dummy-database which is used to speed up the tests of the manager classes.

After setting up the new database and database connection, the adding of a connection string³ was necessary. This caused the problem described in section 5.3.1. Besides the changes done to the database connection, new properties were added to the database. The new database design can be seen in Figure 3.6.

To support using the system on different computers and still having access to the same data, the database needs to be exported and imported (see section 3.4.9).

³The connection string consists of a relative path to the database.

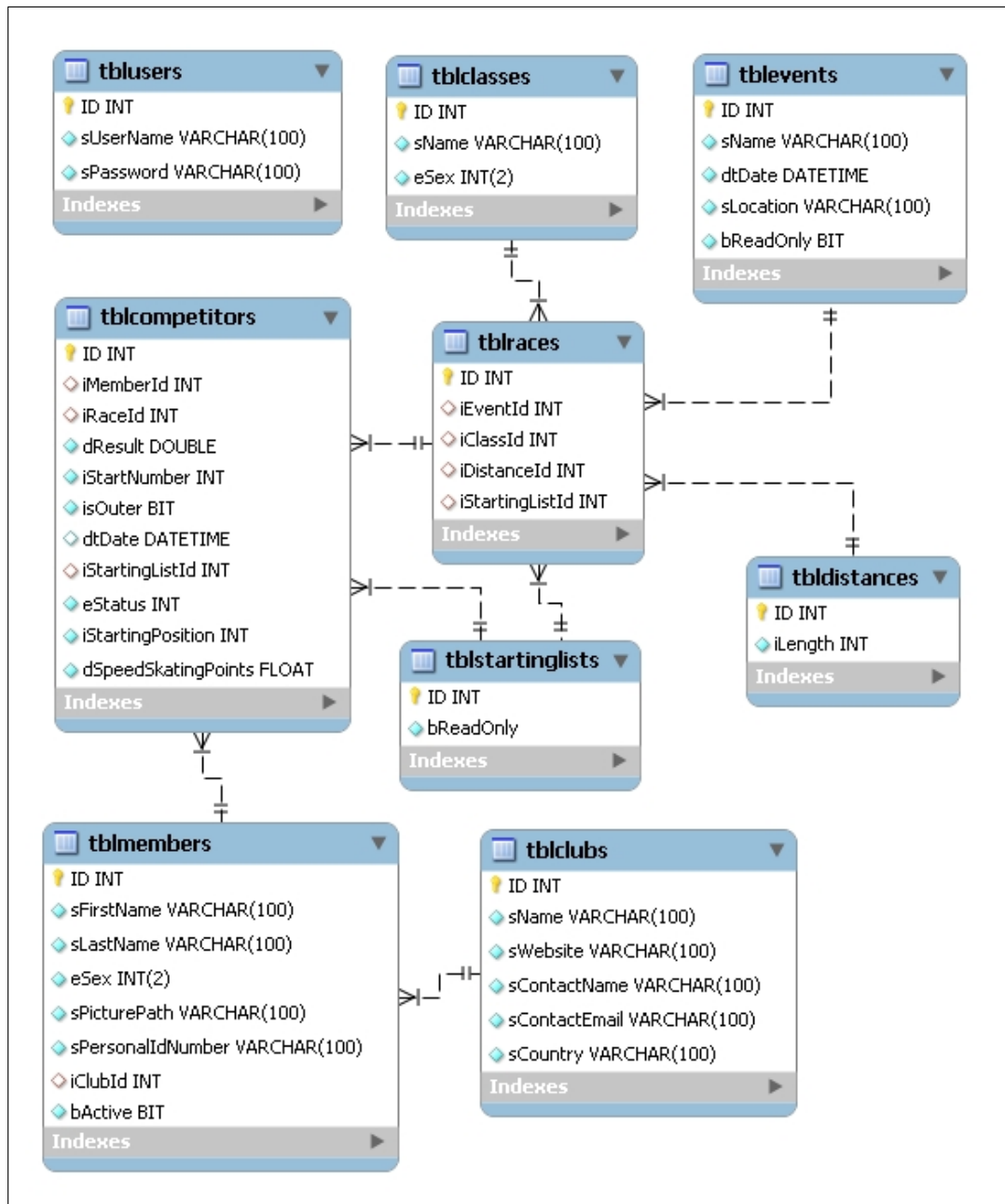


Figure 3.6: Design of the new database.

Create a new user interface

To create a working desktop application, building a new user interface was necessary. Due to the user interface being a very big and important part of the system, chapter 4 is devoted to it.

3.4.6 Skaters, Events and Clubs

This section covers the changes done to parts of the system that represent skaters, events and clubs. Those changes included adding several columns to the database. The database structure after adding the columns can be seen in Figure 3.6.

Add four distances to a class

When setting up the data for an event, each class is registered with four distances. The old user interface provided the possibility to choose exactly four distances from a list of all distances in the system. This function was implemented by displaying the list with a checkbox for each distance. It was not possible to choose a distance more than one time, like it was requested by the customer. Therefore, the new user interface provides four drop-down list to select the four distances registered with a class in an event (see section 4.3.3).

Add a personal best, not connected to an event managed by the application

Every skater in the system can have times for personal bests. There are two possibilities. On the one hand, the skater's personal best is automatically set if it was achieved within an event managed by the application. The other possibility is that the user can set a skater's personal best manually. This has to be done when a skater got a personal best at a race that was not managed within the application. The user can select the distance and simply type in the new personal best time. If the system does not already contain a better time

for this skater the new personal best is added. If there is already a better time, an error message is displayed.

Set a skater active or inactive

Each skater retires someday and therefore does not take part in events anymore. To reduce the amount of skaters shown, when the user wants to add skaters to an event, the inactive skaters are filtered. To support this functionality a new column needed to be added to the skaters' database table and the container representing a skater in the application needed to be updated. At this point the problem described in section 5.3.3 occurred.

Set an event to current or previous

Several weeks after an event has taken place all the results are locked and therefore cannot be changed anymore. To let the application reflect this, a new column was added to the database, supporting the setting of events to current or previous. A current event, its starting lists, results and competitors can be changed. After an event is set to previous no information in it can be edited. The starting and result list, as well as the list of registered classes with their belonging distances, can always be displayed and do not depend on the current/previous status of the event.

Show the number of skaters participating in one class of an event

To help the user to see if she or he forgot to add one or more skaters to a class in an event, the list of all classes registered in an event should also contain the number of registered skaters. To achieve this, only small changes in a container class and its belonging tests needed to be done.

Add country to club information

Showing the country of a skater in the starting or result list is a very important feature. Based on the information the customer gave us, that a club only consists of skaters of one nationality, we decided to save the country in the club. Therefore, we needed to add a column to the club's database table, revise its container and update all the belonging tests. This again caused the problem described in section 5.3.3.

3.4.7 User manual

To make sure, that the user can use the program with all its given functionality correctly and efficiently we also wrote a user manual. The user manual can be found as Appendix A.2.

3.4.8 Edit and remove

To use the program properly it is necessary that stored information can be edited or removed. Deleting information from the database is complicated because everything within the application is connected. For example, an event can only be deleted, if all connected results and races are deleted as well, because they cannot exist without an event. To support this a lot of validation needed to be implemented and tested. After creating the new methods, the user can delete classes, distances, events, clubs, skaters, competitors and personal bests not connected to an event. For editing information in events, clubs and skaters new methods needed to be implemented and tested. Implementing this kind of functionality required adding new methods to all of the three layers. Also new tests needed to be added for all the existing database connections.

3.4.9 Export and import of the database

The application will be used on more than one computer. To support that all instances of the application can work on the same data, the database had to be exported and imported again. In this context, exporting the database stands for copying the database file from the working directory to a directory the user can specify during the export. Import stands for replacing the current database file used by the application with another one. This functionality does not provide migration of the database or exporting or importing of specific parts of the database.

Export the database

The database export consists of retrieving the current location of the database, opening a file dialog and copying the currently used database to the path selected using the file dialog.

Import the database

The import of the database was more complex than the export functionality, because all open database connections needed to be closed first (see section 5.3.5). To dispose the mappers and by this closing all open database connections, a revision of the mapper classes was necessary. Thereafter, the old database file could be replaced by a new one, which was previously selected using a file dialog. Finally, new instances of the mappers were created to re-establish the connection between the application and the database.

3.4.10 Publishing the application

The customer wanted us to create a setup file, so that the program can be installed on the computer. To create this setup file the application needed to be published using Microsoft Visual Studio. While solving this task the problem described in section 5.3.6 occurred.

3.5 Summary

This chapter described the product development. We also discussed the tools and languages we used including the for us very important and helpful technique called Scrum. We structured each sprint and met the customer afterwards to show him, what we achieved. This chapter also covered the changes we have done to the system. A new database and database connection was developed. Within the 'Domain Logic Layer' we added many new methods and a lot of functionality concerning the speed skating points, the starting and the result list, export and import of the database and the export to Microsoft Excel, which was one of the most important parts for the customer. In addition we created a totally new user interface. Due to this interface being a new and big part of the project, we decided it is worth a chapter of its own.

Chapter 4

The new User Interface

This chapter describes the user interface, including the structure, the layout and a detailed description of all the pages. It also explains why the interface is designed like it is. The old interface was web based, so a new interface was necessary after changing the program to a desktop application. The new user interface was created using WPF (see section 3.2.2).

4.1 Structure and Navigation

We tried to follow the advice given by Joel Spolsky: "A user interface is well-designed when the program behaves exactly how the user thought it would." [7]. This is not always easy, because as a programmer you see everything with different eyes than the user does. That is why we did some testing and gave the nearly ready program to the customer for evaluation, which turned out to be very useful. Finally, we came up with the structure shown in Figure 4.1. Individual screens in the program are described by separate rectangles.

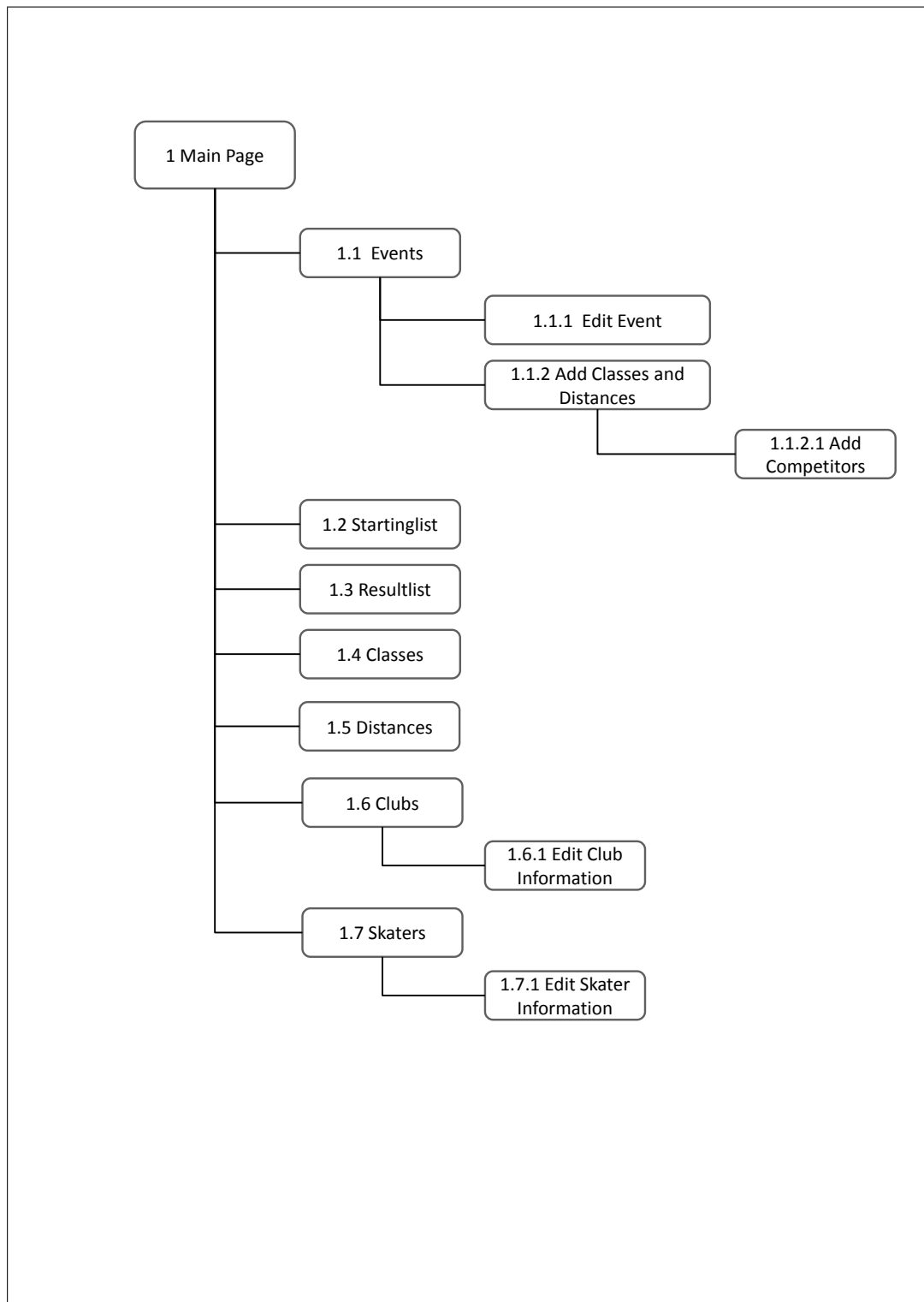


Figure 4.1: Structure of the new user interface.

4.2 Layout

With a simple and minimalistic, but helpful and supporting layout the user should feel comfortable when using the program. To separate parts from each other, coloured lines are used. Important content is displayed in coloured rectangles. The main colour is white, but special separators are displayed in light blue. The colours fit to winter and especially to ice and snow, which is always part of speed skating events. Knowing that users do not always read manuals, everything should be as self-explaining and intuitive as possible.

4.3 Page Description

In the following, all pages of the user interface are described in relation to the handling for the user.

4.3.1 Main Page

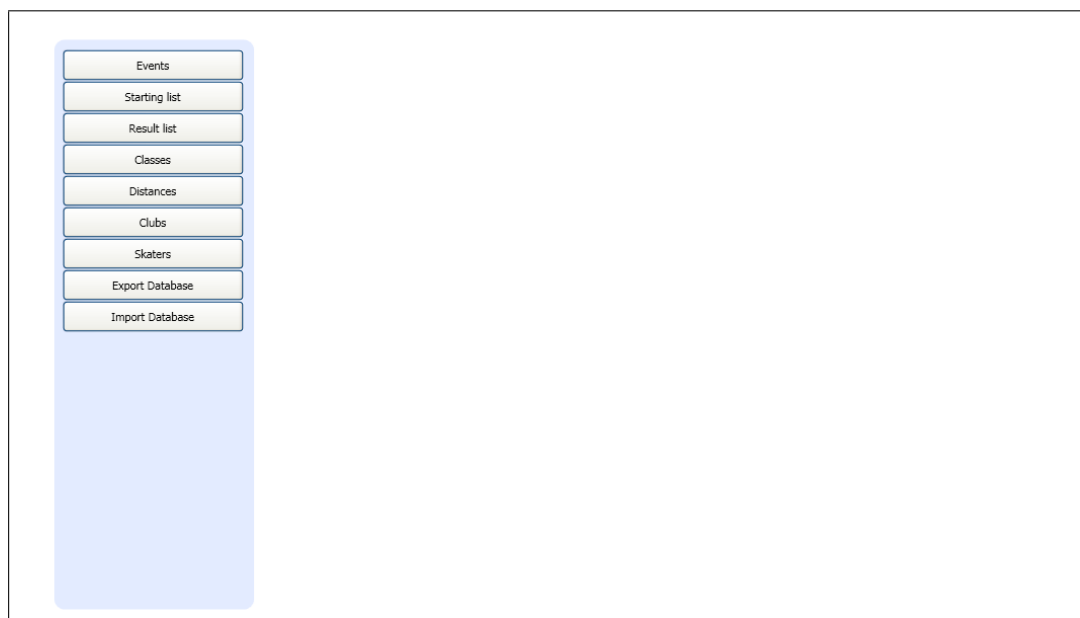


Figure 4.2: The user interface of the main page.

The main page is the first page the user sees after starting the program (see Figure 4.2). On the left side a navigation bar with several buttons is displayed. The bar is in a slightly coloured rectangle to separate it from the rest of the view. If the user presses a button, she or he is redirected to a page, which appears on the right side. During the whole session the navigation bar is always visible.

4.3.2 Events

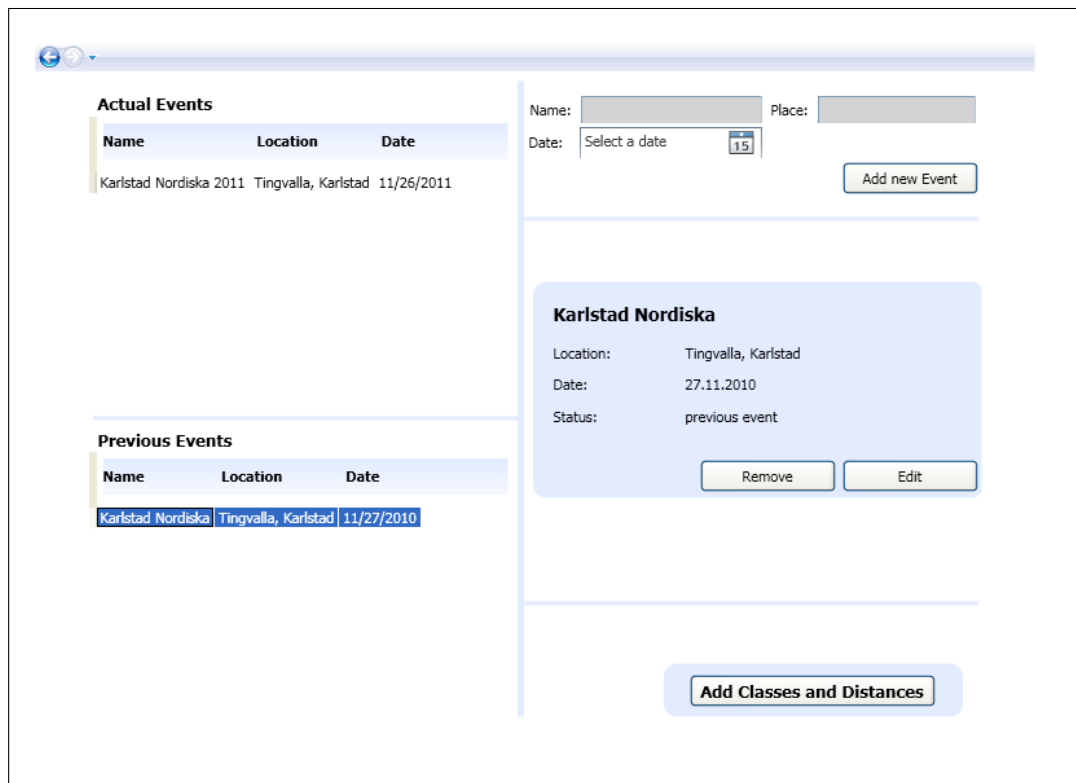
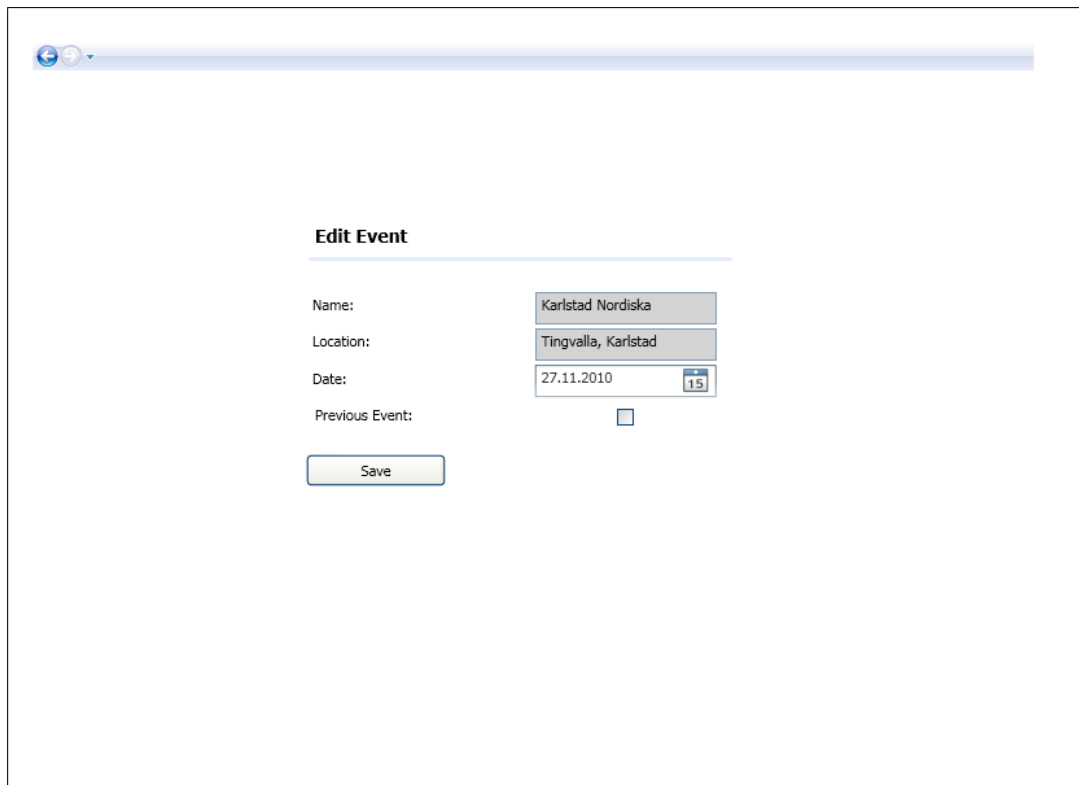


Figure 4.3: The user interface of the events page.

The main functionality of the program is to organize an event. That's why the first linked page is the page for handling events (see Figure 4.3). The page is divided into two parts. On the lefthand side the user can see actual events and previous events. She or he has an overview over all events in the system. On the righthand side the user can

add new events to the system and see detailed information about a selected event on the left. The events are displayed with name, location and date. It is also possible to edit the information of an event (see section 4.3.3). After selecting an event the user is supposed to go to the next page, which is called 'Add Classes and Distances' (see section 4.3.4). This can be achieved by double-clicking the event in the column of the actual events or by pressing the button 'Add Classes and Distances'.

4.3.3 Edit Event



The screenshot shows a web browser window with a title bar. The main content area is titled "Edit Event" and contains a form with the following fields:

- Name: Karlstad Nordiska
- Location: Tingvalla, Karlstad
- Date: 27.11.2010 (with a calendar icon and the number 15)
- Previous Event:

At the bottom of the form is a "Save" button.

Figure 4.4: The user interface for editing an event's information.

The 'Edit Event' page is for editing events and is very simple (see Figure 4.4). The old name, location and date are displayed and are easy to change in here. The user also has the possibility to set the event to a previous event. After clicking on the Save-Button, the

below.

4.3.5 Add Competitors

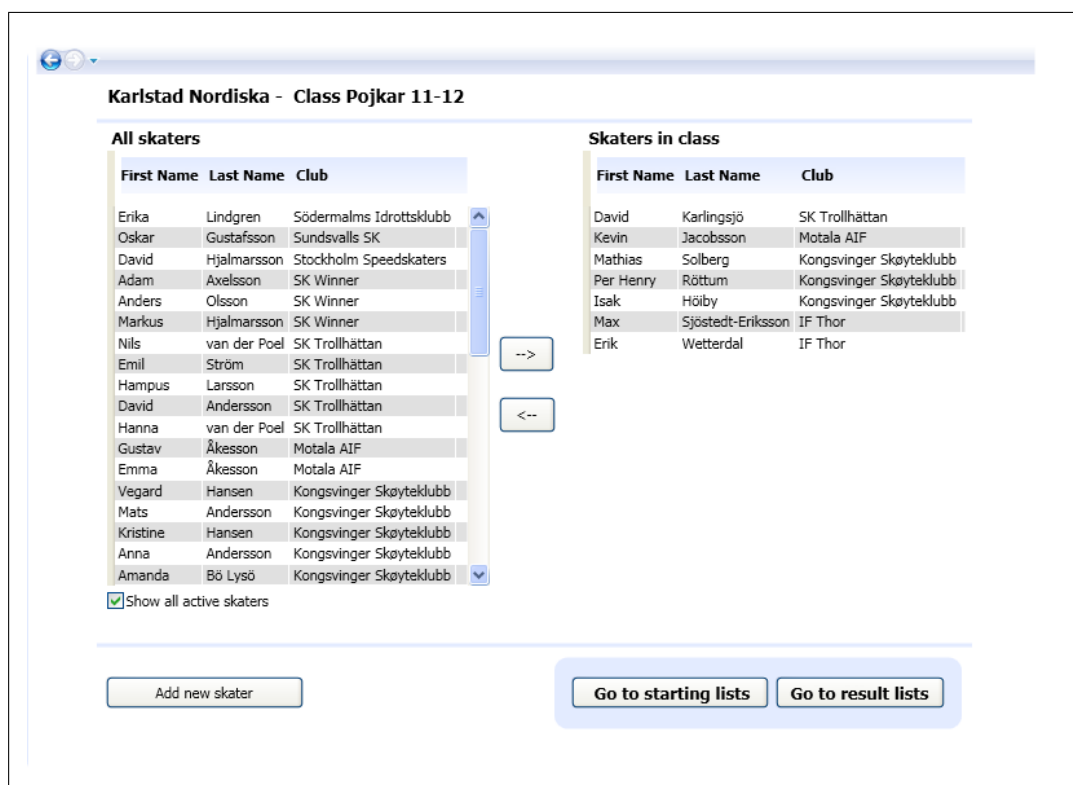


Figure 4.6: The user interface for adding skaters to an event.

On the 'Add Competitors' page (see Figure 4.6) the user can add skaters to a class in an event. The page shows a list of all skaters on the lefthand side and the list of skaters, which are registered in the class on the righthand side. In between these lists there are two arrows pointing in contrary directions. To add one or several skaters to the class, they have to be selected in the list to the left and the button with an arrow pointing to the right list must be clicked. Then the skaters disappear from the list of all skaters and appear in the list of the skaters in the class. To remove a skater from the class, she or he has to be selected in the right list and the button with the arrow pointing to the left list must be

clicked. To make the selection easier, the skaters are filtered according to the gender of the class. If it is a male-class, only male skaters are shown in the list of all skaters. However, sometimes it can happen that for example a woman races in a male class. Therefore a little checkbox can be enabled to show all active skaters in the system. If the user misses a skater in the list she or he can add a new one and is, after clicking on the "Add new Skater" button, redirected to the Skaters-Page (see section 4.3.11). Additional skaters, which are racing in other classes of the event, are not displayed. Therefore, it is not possible to add a skater twice by accident. However, it is still sometimes the case that a skater should be added to two classes in an event. The little checkbox can be enabled again and in this case all skaters are displayed.

4.3.6 Starting List

The 'Starting list' page provides the possibility to create a starting list for an event, a class and a connected distance (see Figure 4.7). All the skaters, which are racing in the class and distance are shown in the list. There are three possibilities to sort the starting list, randomly, after personal best and after ranking (see section 3.4.2). Additionally, the user has the possibility to move the skaters up and down on his own. To do this she or he can use the up and down buttons on the right or right-click the mouse. We wanted to provide more than just one possibility to make changes, so that the user can choose his preferred one.

After sorting the starting list the right way, the user locks the list with a button that is located on the right (see Figure 4.8). Afterwards, a new functionality appears on the bottom of the page and the sorting functionality disappears. Now the user can store the results of the race. Therefore, the first skater in the list (with starting position one) is preselected and a result and status can be added for her or him. The user can select the skater she or he wants, too. After saving, the auto selection goes down to the next skater, so that a fast entering of the results is possible without clicking too much. The user can

Event: Class: Distance:

Starting List - Karlstad Nordiska - Pojkar 11-12 - Distance 1 - 500m

Pair	Lane	No.	Name	Country	Club	Result	Status	Personal Best
1	O	5	Mathias Solberg	NOR	Kongsvinger Skøyteklubb	00:00:00	Pending	00:59:74
2	I	2	Max Sjöstedt-Eriksson	SWE	IF Thor	00:00:00	Pending	01:00:07
2	O	7	David Karlingsjö	SWE	SK Trollhättan	00:00:00	Pending	00:53:92
3	I	3	Isak Höiby	NOR	Kongsvinger Skøyteklubb	00:00:00	Pending	00:53:92
3	O	4	Per Henry Röttum	NOR	Kongsvinger Skøyteklubb	00:00:00	Pending	01:19:79
4	I	6	Kevin Jacobsson	SWE	Motala AIF	00:00:00	Pending	01:10:45
4	O	1	Erik Wetterdal	SWE	IF Thor	00:00:00	Pending	01:25:78

Way of Sorting:

Figure 4.7: The starting list in the unlocked modus.

unlock the starting list as long as there are no results in the list. The reason for this is that the starting list should not be sorted again after the first results are entered. If the race is over the user can go to the page of the result list.

The 'Starting list'-page also provides an export functionality, which exports the list into an Excel file (see section 3.4.4). The user has two possibilities here. She or he can export the list without any results or export it including results and status.

Event: Class: Distance:

Starting List - Karlstad Nordiska - Pojkar 11-12 - Distance 1 - 500m

Pair	Lane	No.	Name	Country	Club	Result	Status	Personal Best
1	O	5	Mathias Solberg	NOR	Kongsvinger Skøyteklubb	01:02:12		
2	I	2	Max Sjöstedt-Eriksson	SWE	IF Thor	01:00:42		01:00:07
2	O	7	David Karlingsjö	SWE	SK Trollhättan	00:55:23		00:53:92
3	I	3	Isak Höiby	NOR	Kongsvinger Skøyteklubb	00:00:00	Disqualified	00:53:92
3	O	4	Per Henry Röttum	NOR	Kongsvinger Skøyteklubb	01:31:04	Fell	01:19:79
4	I	6	Kevin Jacobsson	SWE	Motala AIF	00:00:00	Pending	01:10:45
4	O	1	Erik Wetterdal	SWE	IF Thor	00:00:00	Pending	01:25:78

Result: Status:

include points and status

Figure 4.8: The starting list in the locked modus.

4.3.7 Result List

The page of the result list displays the results of one race in an event (see Figure 4.9). The list is automatically sorted ascending according to the speed skating points. As well as the 'Starting list'-page, the 'Result list'-page provides export to Microsoft Excel functionality (see chapter 3.4.4). With the help of a checkbox the user can choose if she or he wants to export the actual class or all classes in the event. Furthermore, the user can use the four checkboxes to select the distances he wants to export.

Event: Karlstad Nordiska Class: Pojkar 11-12 Distance: Distance 1

Result list - Karlstad Nordiska - Pojkar 11-12 - Distance 1 - 500m

Rank	No.	Name	Country	Club	Pair	Lane	Result	Points	Status
1	7	David Karlingsjö	SWE	SK Trollhättan	2	O	00:55:23	55.23	
2	2	Max Sjöstedt-Eriksson	SWE	IF Thor	2	I	01:00:42	60.42	
3	5	Mathias Solberg	NOR	Kongsvinger Skøyteklubb	1	O	01:02:12	62.12	
4	4	Per Henry Röttum	NOR	Kongsvinger Skøyteklubb	3	O	01:31:04	91.04	Fell
5	1	Erik Wetterdal	SWE	IF Thor	4	O	00:00:00	0	Pending
6	6	Kevin Jacobsson	SWE	Motala AIF	4	I	00:00:00	0	Pending
7	3	Isak Hölby	NOR	Kongsvinger Skøyteklubb	3	I	00:00:00	0	Disqualified

Distance 1 Distance 2 Distance 3 Distance 4 all Classes

Figure 4.9: The user interface for the result list.

4.3.8 Classes

The page for classes gives an overview of all classes that are available. In the table with the classes, the belonging gender is displayed as well (see Figure 4.10). The most important functionality of this page is to add new classes to the system, which can be used in events.

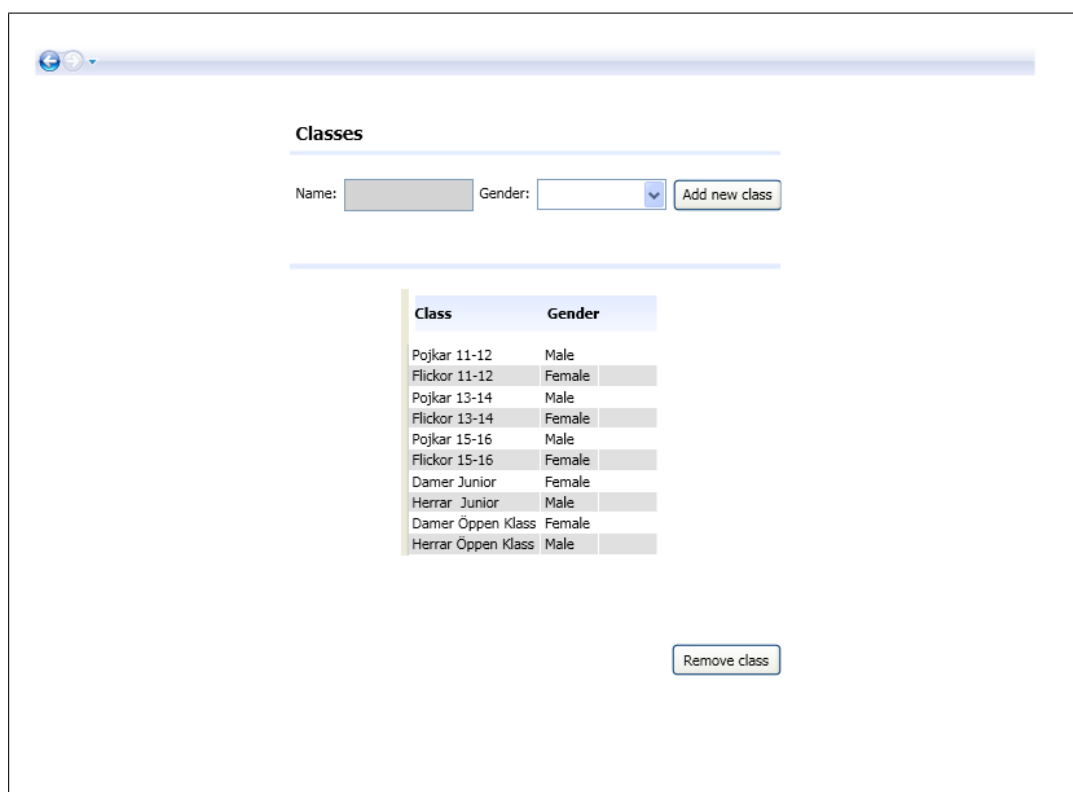


Figure 4.10: The user interface for adding classes.

4.3.9 Distances

The layout of the page for distances is similar to the page for classes. On this page all distances are shown and new distances can be added to the system.

4.3.10 Clubs

The page for clubs provides functionality for administrating the clubs (see Figure 4.11). The page is divided into sections to have a clear arranged layout, which makes the handling easier for the user. At the right of the table there is a vertical line which separates the page optically into two halves. The overview is placed on the left side and the detailed view and administration part is placed on the right side. The detailed information is located in a slightly coloured rectangle, which encircles the information and separates it from the rest

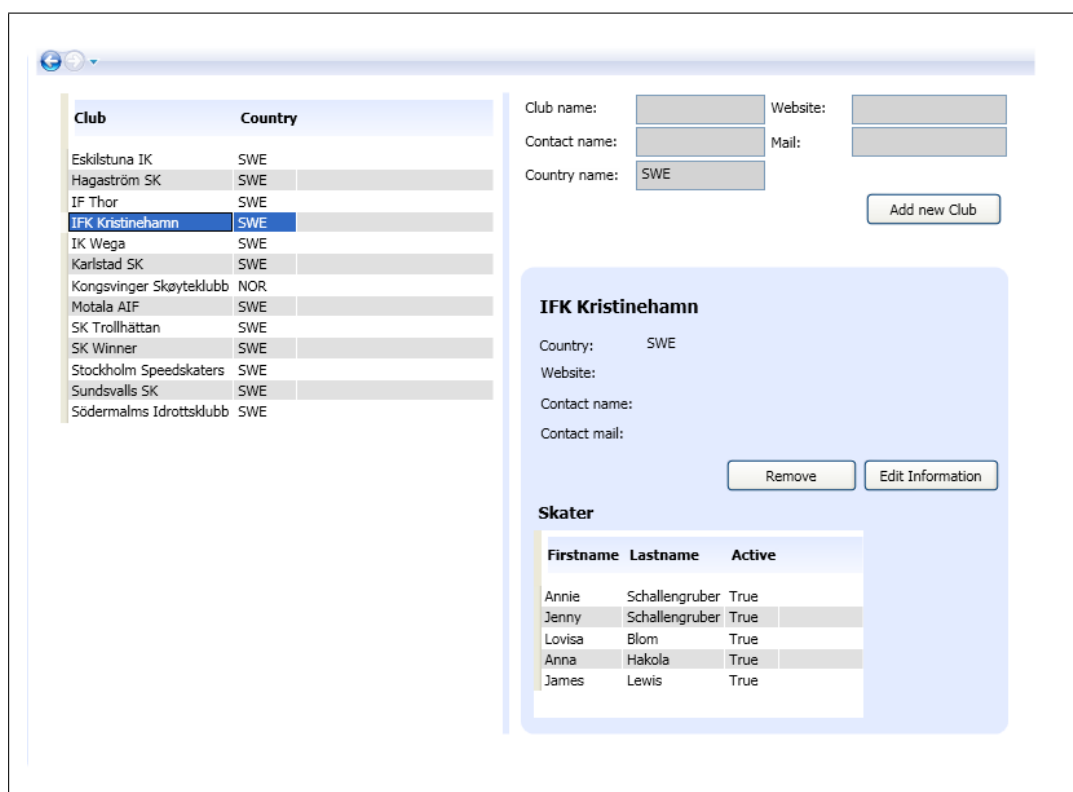


Figure 4.11: The user interface for the club page.

of the page. The club page provides the possibility of editing club information. If the user presses a button she or he is redirected to another page, whose layout is of a minimalistic design and does not disturb the user. On this page the user can edit all the information she or he wants and save it. Afterwards, she or he is navigated back to the club page.

4.3.11 Skaters

The layout of the 'Skater'-page is the same as the layout of the 'Club'-page (see Figure 4.12). For the user this is easier to handle and she or he does not have to adapt to a new environment. The only differences from the 'Club'-page are the contents. In the grid on the left side all skaters in the system are displayed with their first name, their last name, their club and their active/passive status. On the upper right side it is possible to add

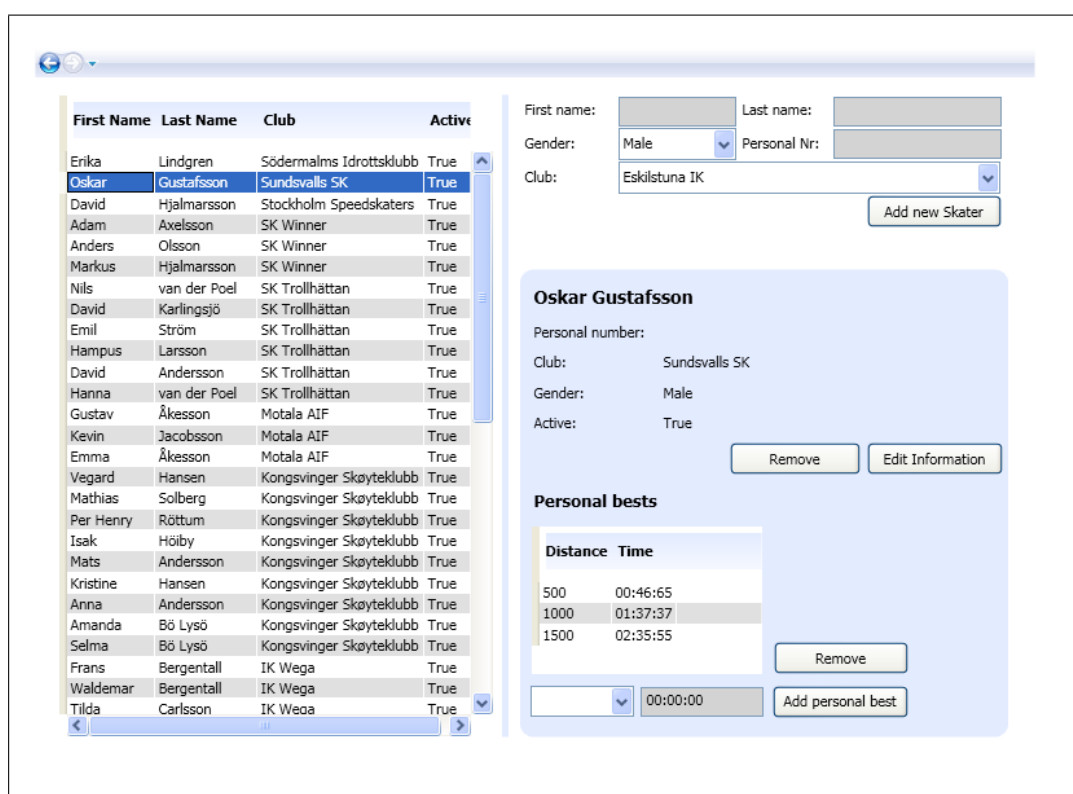


Figure 4.12: The user interface of the skaters page.

a new skater, who appears in the list on the left side after clicking the 'Add new Skater' button. On the lower right side the details of the selected skater are shown, including their personal bests for each distance. To edit information about the skater the user can press a button and is redirected to another page. On this page the user can edit all the information about the skater. After saving she or he is navigated back to the 'Skater'-page.

The user can add new personal bests, which are not connected to an event in the system. To do so the user needs to select a distance in the drop-down-list, type in the time and click the button 'Add personal best'. If the new personal best is better than the current one, it appears in the skater's personal best table.

4.4 Summary

This chapter covered a description user interface's navigation, layout and a detailed description of the different pages of the user.

Chapter 5

Results and Evaluation

This chapter contains the comparison between the tasks we set ourselves to do in the beginning of the project in relation to the tasks we solved. It also covers a description of the problems we faced and a brief explanation about how we solved them.

5.1 Results

Our task was to implement a program, which would help the customer to organize speed skating events. As a basis we had a given system implemented by a group of students. This system was a server application that had a web based user interface, which the customer requested at this time. At our meetings we found out that there is no constant internet connection at the location the races take place. Therefore, we had to think about a new solution. We finally decided to create a desktop application for the customer. The system got a new user interface, database and database connection, but by obeying the three layer architecture (see section 3.2.1), it is still possible to reconvert the system to a server application using the same domain logic.

Although we had a given system we had to change and add a lot. We chose Scrum (see section 3.3.2) as the development model and worked in two week sprints. Each sprint

finished with a meeting with the customer. Therefore, it was possible to let the customer take part in the developing process. This turned out to be very useful, because a lot of misunderstandings could be resolved at the meetings. At each meeting the system seemed to grow more and more. However, the reason for this was that the customer noticed that some details were missing, wrong or that some additional functionality should be added. For example, we were not aware that a race can have the same distance more than one time. So we implemented everything we should and were happy about it. But when presenting it to the customer, he told us about the mistake. If we had not met him frequently, we would never have noticed the mistake.

We implemented all the functionality left from Table 2.2, which means that we completed the basic functionality for an application managing speed skating events. This included completing the starting list and the result list. Speed skating point calculation, active/passive state of a skater, actual/previous state of an event, proper adding of distances while registering a class in an event and edit/remove functionality were implemented as well. Furthermore, we implemented additional functionality that was important for the customer, such as conversion to a desktop application, export to Microsoft Excel, import/export of the database as well as including additional information for skaters, events and clubs.

Even though we had many problems during the implementation process it was still possible to create an application, which has a high value for the customer.

5.2 Beta Testing

After developing a usable program, we handed it to the customer for testing. The testing turned out to be very useful since a number of problems with the user interface were discovered. The user interface was developed in Microsoft Visual Studio running on operating system Windows XP. We did not expect that the user interface would look different in

Windows 7. The beta test showed us that the components seem to be smaller in Windows 7 and the font size bigger. The result was that the customer could not read everything, because some labels were too small (see Figure 5.1).

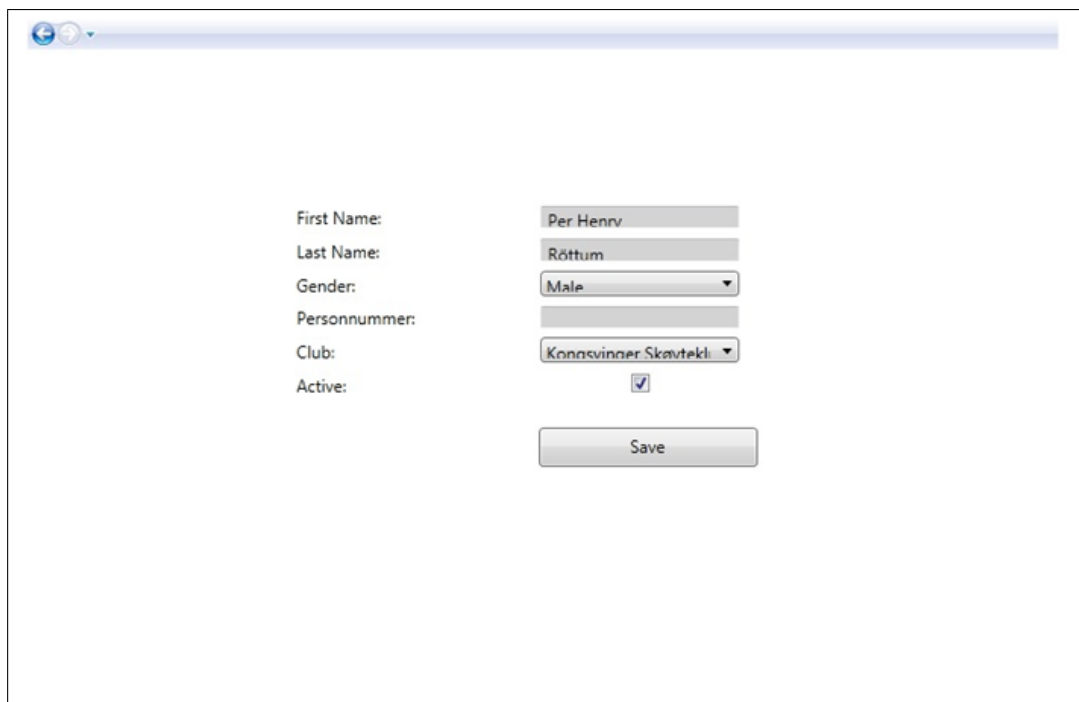
A screenshot of a web application form in a Windows 7 window. The form contains the following fields: 'First Name' with the value 'Per Henv', 'Last Name' with the value 'Röttum', 'Gender' with a dropdown menu set to 'Male', 'Personnummer' with an empty text input field, 'Club' with a dropdown menu set to 'Könnsvinger Skåvtekl', and 'Active' with a checked checkbox. A 'Save' button is located at the bottom of the form.

Figure 5.1: User Interface Problems in Windows 7

Another problem was, that the customer simply overlooked some buttons and thus had problems with the navigation. Therefore, we decided to highlight them a bit more. Additionally, we had to explain some of the functionality to the customer and noticed that a user manual would be very helpful. After the beta testing, the database was filled and we noticed that some tables were too small for displaying all the necessary information. Therefore we decided to make their size bigger.

All in all our customer was satisfied with the program.

5.3 Problems

This sections cover the major problems, which occurred during our work on the project and our solutions for them.

5.3.1 Relative connection string

One of the first bigger problems that occurred was that the application crashed during testing. This happened due to the new database being stored locally and because we used a connection string based on a relative path. The database laid in the project containing the code for the 'Domain Logic Layer' and the 'Data Access Layer'. The tests were part of a separate project. Both projects needed access to the database. Therefore, the connection string, consisting of a relative path to the database, needed to be adapted depending on the project, which was executed. A class generating the connection string according to the executed project solved this problem. Later on, we needed to change this class again, because we moved the database to a different project, the user interface (see section 5.3.6).

5.3.2 Database connections

Too many open database connections made the application crash as well.

The problem occurred because we used the wrong way of accessing the mappers from the manager classes. As seen in Figure 5.2 the managers accessed the mappers by using the factory class. Creating a manager class caused the instantiation of a mapper class. Additionally, the manager held instances of several other manager classes to gain access to other database tables and therefore the possibility to perform more complex operations. Because the manager classes were instantiating each other and also several mapper classes, the number of open database connections increased very fast while running the program. This mistake lead inevitable to a crash of the application and could only be solved by redesigning the manager classes. The redesigned manager now holds only instances of

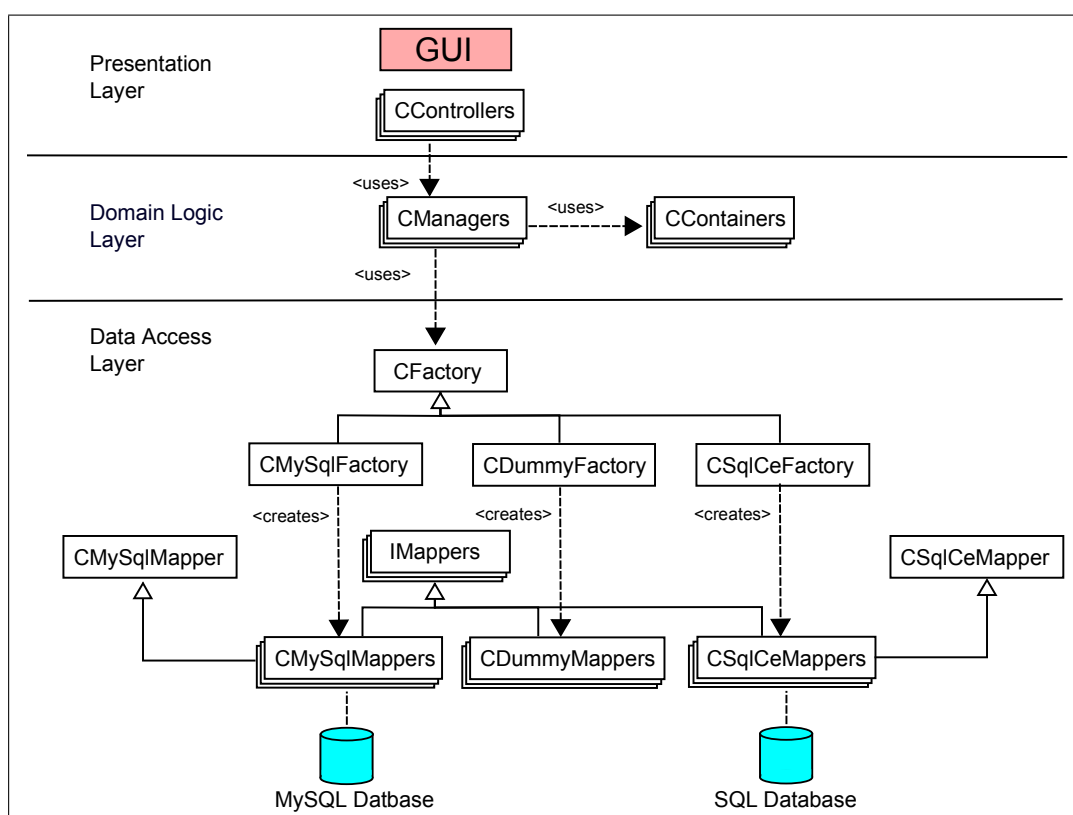


Figure 5.2: Reduced class diagram of the new system

mapper classes. This violates the separation of concerns, but was the only way to prevent the problem with the manager classes instantiating each other. To reduce the database connections further, the mapper classes were not instantiated when creating a manager class, but at the start of the application. Whenever a manager class is created, it retrieves an already existing instance of a mapper class.

5.3.3 Adding new columns to the tables of the databases

To provide more information for starting and result lists, clubs, events and skaters we added new columns to the tables of the databases, updated the mappers and needed to edit the container classes. The updates were done to all databases and database connections. The result was that nearly all tests failed, because the constructors of the container classes

had changed. To solve this problem nearly all tests needed to be updated manually. Unfortunately, the changes regarding adding database columns came up several times and we had to update the tests often.

5.3.4 Data bindings

Due to data bindings in WPF (see section 3.2.2) it was hard to display additional information not contained in a container class¹. For example, how many skaters are registered in a class in an event. These data bindings to an object always required the design of a new container, which could be used for displaying additional information together with the regular information. In the example with the registered skaters a new container class, including the class registered to an event, the four distances assigned to the class and the number of skaters, needed to be created.

5.3.5 Importing a new database

When we implemented the import of a new database we figured out that the old database file could not be overwritten. The reason for this was that the database was locked because of open database connections in the mappers. The changes done in section 5.3.2 were beneficial when solving this problem, because it was easy to retrieve all instances of mappers. Before importing, all instances of mappers had to be disposed of, which closed all open database connections. Afterwards the database could be replaced. Finally, new instances of the mappers were created.

5.3.6 Publishing the application

As the customer requested, the application should be published as a setup file. After running the setup file the program should be installed and ready to be used. At this point

¹A container class is a digital representation, containing all necessary information concerning for example a skater.

the installation of the program threw an exception and was not usable at all. By searching on the internet and trying out several different settings to create the setup file, we found out that the database was not included in the setup file. This problem occurred because our solution was structured in different projects according to the three layer architecture (see section 3.2.1), where the 'Domain Logic Layer' as well as the 'Data Access Layer' shared a project and the 'Presentation Layer' was a project on its own. Because the database was not located in the startup project, Microsoft Visual Studio had a problem including all necessary files. This problem could be solved by moving the database to the startup project.

The connection string for the database caused problems as well. After moving it to the user interface, we needed to reconfigure the class, converting the connection string depending on the executed project (see section 5.3.1). Then the application could be published as a setup file, installed using it and executed without any further problems.

5.4 Summary

This section gave a brief overview of the development of the application and about which problems we faced. It also contained the results from the beta test and a short evaluation of the program and its functionality.

Chapter 6

Conclusion and Future Work

After working with this project for half a year we are very satisfied with the results. We implemented all the basic requirements, were able to complete the list of tasks necessary for an application for managing speed skating events and added even more functionality. During the meetings with the customer he showed us that he is very pleased with the result.

We were very optimistic at the beginning of the project and thought we would be able to solve the basic requirements and could at least start with a part of the additional requirements. In the end we implemented only the basic requirements, which was still a lot of work.

We used Scrum and tried to meet our customer as often as possible. During those meetings we always discussed the requirements and moved tasks from the basic requirements to the additional requirements and vice versa. By using Scrum we developed the application the customer wanted and prevented mistakes based on wrong interpretation of tasks.

6.1 Experiences

We had several very time consuming tasks. One of them was, as can be read in section 5.3.3, that each change made to the database resulted in revising all the tests, big parts of the mappers and containers. We could have prevented the extra work, if we had made all changes at once. However, as written in section 2.3 we constantly negotiated with the customer, which often resulted in adding new columns to the database. That is probably one of the main reasons why we had to add new columns and revise everything several times.

We could have started to work on the dissertation earlier and spent more time on it. However, because we wanted to deliver a program, which has a high value for our customer, we made the mistake of programming too much instead of working on our dissertation. This bettered after a short time, but it would still have been better, if we had started earlier.

Maybe we should have focused more on one task, instead of doing several tasks in parallel. Because of the parallel working we sometimes did not test all the edge cases and therefore had some problems with errors.

6.2 Future work

As already mentioned above, we solved the basic requirements shown in Table 2.3. The additional requirements (see Table 6.1) are still not implemented and could be done later on.

Besides the tasks mentioned above there are still a lot of other parts of functionality, which could be added. Three examples are given below:

Inclusion of a time tracking system

The Karlstad Speed Skating Club has a time tracking system, which is, until this point,

Additional requirement	Description
1. Starting list 2. Online Application 3. Starting Fee 4. Relay 5. Provide informationl	<ul style="list-style-type: none"> - Provide withdraw functionality - Reduce the functionality of the online application to managing clubs and skaters. Provide functionality to register a skater to a class in an event. - Each club should get a user name and password to manage his skaters, information and registration for the event on his one. - Provide functionality to manage the starting fees for an event - Include relay functionalities - Add phone number, country, city, club logo to a club - Add website, Facebook, Twitter, Email, phone number to a skater - Add photo of the arena to the event

Table 6.1: The tasks still left to do

a program on its own and runs on a very old MAC system. This tracking system could be included in the application, so that the skaters' results are entered automatically.

Serverside program

The program could also be converted to a server application so that the clubs can register their skaters themself in the system and for events.

Migration of database

The database import and export could be reworked so that the new database and the old one are migrated instead of the new one overwriting the old database.

References

- [1] Henrik Kniberg. *Scrum and XP from the Trenches*. InfoQ Enterprise Software Development Series, 2007.
- [2] John C. Kunz. Concurrent knowledge systems engineering - working paper number 5, July. Website, last checked: 2011-05-09, url: <http://www-leland.stanford.edu/group/CIFE/online.publications/WP005.pdf>.
- [3] Microsoft. Sql server compact 3.5, 2011. Website, last checked: 2011-04-01, url: <http://www.microsoft.com/sqlserver/2005/en/us/compact.aspx>.
- [4] Dr. Winston W. Royce. Managing the development of large software systems, 1970. Website, last checked: 2011-05-01, url: <http://www.cs.umd.edu/class/spring2003/cmcs838p/Process/waterfall.pdf>.
- [5] Chriss Sells and Ian Griffiths. *Progammimg WPF - Building Windows UI with Windows Presentation Foundation*. O'Reilly, 2nd edition, 2007.
- [6] Sheldonleecooper. C sharp (programming language), 2011. Website, last checked: 2011-03-09, url: http://en.wikipedia.org/w/index.php?title=C_Sharp_%28programming_language%29&oldid=415675308.
- [7] Joel Spolsky. *User Interface Design For Programmers*. Berlin: Springer, 2001.