FAKULTÄT FÜR !NFORMATIK

Usability Enhanced Agile Development for Web-Based Applications

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Magister der Sozial- und Wirtschaftswissenschaften

im Rahmen des Studiums

Informatikmanagement

ausgeführt von

Friedrich Dimmel, Bakk.techn.
Matrikelnummer 0302230

am:
Institut für Knowledge and Business Engineering, Universität Wien

Betreuung:
Betreuer/Betreuerin: a.o.Univ.-Prof. Dipl.-Ing. Dr. Renate Motschnig

Wien, 15.10.2008

_____ (Unterschrift Verfasser/in)

_____ (Unterschrift Betreuer/in)

I assure that I wrote this diploma thesis myself and I only used the stated sources and tools.

Ich versichere, dass ich diese Diplomarbeit selbstständig verfasst, und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Vienna, October 2008

Acknowledgement

Before going *in medias res*, I want to thank those people that supported me during the past years of my life, especially during my studies.

First of all I want to thank my parents, Margarete and Friedrich, which always stood and hopefully will be standing behind me. Without their love and helpfulness, I wouldn't have been able to be who I am and to have the luxury to finish a study. I want to thank them for their patience and generousness and finally their love towards me.

I also want to thank the rest of my family for their endless support in good and bad times.

I want to thank Klara, for her never-ending love and her support and motivation when writing this thesis.

Thanks to Günter Baumgartner and Peter Trimmel for their great mentoring and professional support, and their English skills when proofreading this thesis. I also want to thank you for letting me learn so much from you and your professional attitude for work.

Thanks to Christoph Kroneder for his insightful presentations and discussions about Agile Methods.

Thanks to Andi Hejl and Georg Kreuch for the budget for implementing the practical part of this thesis.

Thanks to Simone Kriglstein for her detailed feedback.

And finally thanks to Prof. Renate Motschnig for her feedback and supervision of this work.

Thank you all for your helpfulness. I wouldn't have been able to do this without you!

Abstract

During the past few years the author has been working at Siemens as a working student with a part-time employment. He worked within the “Support Center for Components & Internet Technology” group, which was responsible to stay up-to-date with modern Internet technologies and be technical leader for web trends. A certain part of the daily work was to examine new technologies, programming languages and hypes, building know-how and implement prototypes for demonstration purposes.

The past years brought lots of new technologies to the web, which finally was called Web 2.0, because of new possibilities for users and developers. One of the latest trends is the use of Rich Internet Applications, which add new functionality to browser-based applications and integrate the efficient usage of rich data sets. Also the integration of rich media and multimedia items is a central, new opportunity since the introduction of Rich Internet Applications.

Many different start-ups evolved and they built user-centric web-applications. Sometimes different sites had the same idea, but only a few grew to huge websites. Because these sites were done by start-ups, which weren’t able to put millions of dollars into marketing, there must be another cause, why some sites were successful and some weren’t. One of the important differences to other sites was most likely usability. Usability for web sites is more and more important when the website’s purpose is the integration of the user and his participation. Different guidelines are available for usability, but most of them are very old. It’s important to adapt these guidelines for modern web applications, and finally Rich Internet Applications.

The author of this thesis selected the topic, because of his interests in web applications, usability and agile development. Also his Bachelor thesis was about “Rich Internet Applications and Service Oriented Architectures”; the thesis at hand dives into a more detailed description of how to implement such applications in terms of better usability and the development process using Agile Methods.

Abstract (German)

Während seiner Studienzeit arbeitete der Autor dieser Arbeit als Werksstudent bei Siemens mit einem Teilzeitvertrag. Er war in dem „Support Center for Components & Internet Technology“ tätig. Der Zuständigkeitsbereich des Support Centers lag darin, neue Technologien, vor allem im Bereich von Internet und Web ausfindig zu machen und als technischer Ansprechpartner für diese Technologien im Konzern kompetent zu sein. Neue Programmiersprachen und Web-Technologien war nur ein Teil der Aufgabe, daneben war es wichtig, Know-how Aufbau zu betreiben und Prototypen mit Hilfe neuer Technologien zu erstellen, um Showcases zu zeigen.

Die letzten Jahre brachten vielen neue Technologien in den Web-Bereich. Der neueste Trend, bzw. die Bezeichnung Web 2.0 steht gerade dafür, dass sich für Benutzer und Entwickler viel getan hat und sich neue Möglichkeiten eröffnen. Ein ebenso aktueller Bereich aus dem Umfeld von Web 2.0 sind sogenannte Rich Internet Applications. RIAs sind Web-Applikationen, die im Browser ablaufen aber neue Funktionalitäten einbringen und somit komplexere Daten visualisieren und verarbeiten können. Multimedia Daten, wie z.B. Videos oder Musik, etc. lassen sich ebenso in Rich Internet Applications einbetten.

Die letzten Jahre brachten viele neue Firmen und Start-Ups auf den Markt, vor allem benutzerzentrierte Web-Applikationen sind derzeit ein Hype. Viele verschiedene Firmen hatten wohl die gleichen Ideen, wenn es um solche Applikationen geht, doch nur wenige schafften es bis an die Spitze. Vor allem Start-Ups ohne viel Kapital können sich jedoch keine millionenteuren Marketing-Auftritte leisten; es muss daher andere Grüne geben, warum Firmen mit ihren Ideen erfolgreich sind. Einer dieser Gründe war höchstwahrscheinlich die Usability, also die möglichst einfache Benutzbarkeit von solchen Services. Wenn es darum geht, Benutzer an Webseiten zu binden und vor allem den Benutzer Teil der Webseite werden zu lassen, in dem er zu den Inhalten beiträgt, ist es wichtiger als je zuvor, die Webseite so zu gestalten, dass der Benutzer sich mit ihr zurechtfindet.

Für Software-Entwicklung gibt es viele verschiedene Richtlinien, jedoch kaum angepasst an moderne Web-Applikationen. Gerade der Umstand, dass der Benut-

zer ins Zentrum der Applikationen tritt, erfordert aber die Analyse solcher Richtlinien für Web-Applikationen und im Speziellen auch für Rich Internet Applikationen.

Der Autor dieser Arbeit hat das Thema aufgrund seines großen Interesses zu den Themen Web-Applikationen, Usability und auch Agile Methoden ausgewählt. Bereits seine Bakkalaureatsarbeit handelte von „Rich Internet Applications and Service Oriented Architectures“.
Diese Arbeit beleuchtet die Implementierung von solchen RIAs, speziell im Hinblick auf Usability und den Entwicklungsprozess mit Agilen Methoden.

Table of contents

Introduction

In today's environment IT companies who are creating customized projects for individual customers have to calculate the costs for each single project individually. Most projects require individual development or customization because of domain-specific needs. IT companies have to compete harder than ever to win project tenders and to get a satisfying margin when getting paid for the project.

With increased size of the projects the calculation for the costs are getting more and more complex too. If project development lasts for several years or even the people working for the project are settled in different countries the calculation process takes much time due to different location- and time-specific costs. For project leaders it is very important to have an overview over different parts of the projects and how expensive they are. There should be a simple way to enter and change different variables, which are part of the calculation and represent important parameters of the calculation. And, of course, theses changed values should affect all the following calculations too and the project leader should see how these new values fit into his calculation.

Web-based applications have become ubiquitous in the last years. Well-known hype-words like "Web 2.0" have been an accelerator to the trend to bring applications to the browser and let the user interact with them online. The latest development in this area is to get away from classic client/server architectures where the browser is only the rendering part of the application (the view) and the server has to do all of the business logic, calculations, etc. There are more and more libraries for web development available, which allow rich user interfaces with client-side logic to not let the server bear the entire payload.

With classic client/server architecture for web-based applications each request to the server requires a complete refresh of the view and the browser has to render the entire page again. Additionally the complete rendering information (the design) has to be transferred from the server to the client at each request. This requires a lot of redundant information exchange and wastes a lot of network traffic bandwidth. Modern web applications make use of libraries to cope up with these

problems. Many web applications use AJAX (**A**synchronous **J**avaScript **A**nd **X**ML) to have a connection to the server. Such applications only transfer pure information (data) from the server to the client and vice-versa. The layout has to be transferred only once (and, of course when layout changes are made due to different application parts). For the rest of the usage of the application only the information that is required at the moment has to be transferred.

There's also another type of web-based applications that take advantage of data-only transfer technologies. Rich Internet Applications (RIA) is the generic term, when one's speaking about web applications. RIAs have a rich user interface with user-centered controls providing a better user experience and they are mostly built on a service-oriented architecture (SOA), which provides the interface to data stores and business logic. AJAX-powered applications are such RIAs. But there are different other technologies that allow a better user experience than standard client/server web applications. For example Microsoft Silverlight and, the technology used in this thesis, Adobe Flash.

Different development processes have affected the past decades of software engineering. Common development strategies were the waterfall model, the rational unified process, extreme programming, etc. One quite new strategy in software engineering is the use of "Agile Methods". This method is based on quick iterations of steps as planning, designing, developing and testing. The goal of these short iterations is to keep the overhead for project planning very small and to have many builds of the application. Those builds don't have the full feature-set of the final application but those features, which are implemented, should be working and should be without bugs. Working with Agile Methods is quite similar to prototyping applications. It's important to have parts of the application finished very quickly without having to do lots of administrative tasks and time-consuming design-phases. One of the different specifications of agile development is called "Scrum". Scrum has a predefined set of roles in a project. The developers, designers, etc. are called "team". A team normally consists of a very small number of people who work together. The "Scrum master" is responsible that the Scrum process is executed correctly. He's also host of Scrum meetings (which are called "sprints"). The "product owner" within the Scrum process is the person, which represents the customer to the team and the Scrum master. The product owner is responsible for the feature-set of the application and observes the project from the

business perspective. Typical iterations last 2-4 weeks. Those iterations are called “sprints”. From the complete feature set for the complete applications small parts are picked out into “sprint backlogs”, which represent the work that has to be done until the sprint is over. After each sprint iteration a sprint meeting is scheduled where the Scrum master and the team reflect the past sprint and improvements for the future are discussed. Additionally to the sprint meetings every day a “Scrum meeting” is scheduled. Here every team member is asked questions about his current status and what his plan for the following day is.

An important part of an application is always its usability. Usability means how the user is able to interact with the application. If the usage is complicated and the user has to read many manuals to understand how to use the application, the usability level is low. But if the application is designed in a way that the user can “learning by doing” or the user even can use the application correct instantly, the usability level is very high. Several repeating steps can improve usability during application development. Representative users (for the target audience) performing representative tasks (tasks which will be in focus for the application’s purpose) should be observed. After they are finished, one has to analyze where the testers had problems and which steps have to be improved (e.g. to improve the performance of the application). There are many different aspects of usability that can be measured and all are important for the quality of the product.

Web based applications

Web based applications can be run in the browser of the client computer and interact with a backend on a server. Typically these kinds of applications are called client/server-based applications.

Client/Server-Based Web Applications

Nowadays most web-based applications are developed based on client/server architecture. These applications are mostly text-based (HTML) and require page refreshes on each interaction with the server. That means, if a user enters a value into a form and wants to save this value to the server, a request is made to the server and the server performs a save operation for this value. Maybe the value that was entered by the user is not valid (e.g. the user should enter a number but he entered a letter), and then the server has to inform the user to enter a valid character. So the server creates a web page with an error message to tell the user to correct his entered value. Therefore the entire web page has to be sent to the user again just to tell him that there was a single wrong value. The user then can enter a correct value and submit the page again to the server. If all values are valid and processed on the server, a response page has to be created to tell the user that his values were saved. Again a complete page with all of its design has to be transferred to the client computer's browser and to be rendered there.

From the browser's perspective of view this is very simple to accomplish. The browser only displays information that comes from the server and does not have to process any logic and therefore does not have any responsibility for the entered data. The server however has to take care on all of the logic. The server checks the validity of the user submitted values, interacts with the backend services (maybe other applications, web services, databases, etc.) and has to generate the design and the pages for the client to display. Those tasks may be very complex tasks for the server and require a lot of processing power. If one imagines thousands of simultaneous requests by clients to a server and the server is responsible for every-

thing (business logic and view) this may be a huge payload to take. Also the experience for the user may suffer from this fact because the page roundtrips may take longer because the server is overloaded causing the downgrade of responsiveness.

Service Oriented Web Applications

In the last years some new or improved methods have emerged to better these drawbacks of the non-intelligent client. Technologies such as AJAX, Adobe Flash or Microsoft Silverlight cope up with the problems that everything has to be rendered and transferred over the network again and again. With service oriented architectures and e.g. web services the client can just exchange data, which is needed for the current task. This can speed up the performance and transaction rates of web applications dramatically.

What is a "Web Service"?

Generally a Web Service is known as a term, which stands for programmatic interfaces made available for application-to-application communication. [W3C]

Engineers use the term web service mostly to describe a remotely accessible component, which allows the "service-consumer" to access functionality from another machine. Normally web services are accessed via HTTP or HTTPS from within web applications. Generally web services don't rely on a specific protocol. But for the purpose of this thesis we stick with web services over HTTP(S). An advantage of web service is that they are loosely coupled (at least by design) and therefore consumers don't have to have knowledge of what's behind the service façade and the services don't change their signature when the underlying logic may change. With web services it's possible to get great interoperability between existing applications, whereas one has to keep in mind that web services are not the solely answer for Enterprise Application Integration (EAI). But web services may help to solve EAI issues. Web services are also designed to be understandable by humans and machines. They are described in an XML based language where the services' operations are defined. The language for the description of an entire web services is known as Web Service Description Language (WSDL) [WSDL]. Within this declaration important information about the usage, required parameters for operations, endpoints and protocols are written down.

SOAP [SOAP] is used as a protocol for web service communication. The SOAP messages (request / response) are encoded in XML to be readable by the computer on one hand and understandable for humans on the other hand. SOAP is very similar to XML-RPC (Remote Procedure Call), which stands for XML message exchange over HTTP. The main differences between SOAP and XML-RPC are that SOAP does not rely on a specific protocol as HTTP as XML-RPC does and another difference is that SOAP contains header and body information within a specified envelope (see: Figure 1).

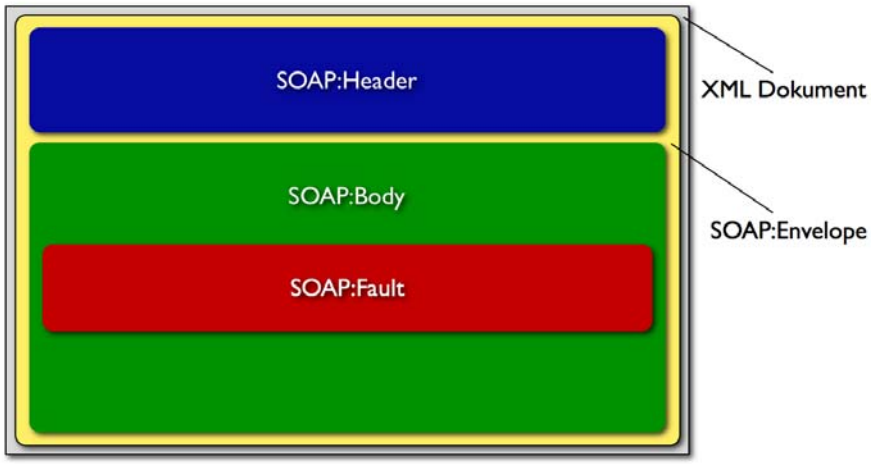

Figure 1: Structure of a SOAP Message

Taking advantage of technologies and architectures such as web services, developers can quickly build web applications that take use of the data and logical operations provided by the services. Examples for such a usage could be to load products for a shopping cart application based on a selected category. At first a list of available product categories is displayed for the user. Then the user selects a category, which invokes a service request to the server. The service then reaches this request further to the internal business logic where a result set is created. This result set then will be passed to the service and the response comes as a SOAP message back to the client. The client application then decodes the SOAP message and creates a list of the available products for the selected category. This can be done with JavaScript libraries (if the interface is based on HTML) or within Adobe Flash or Microsoft Silverlight with their respective scripting languages.

That way would save the HTML overhead for the formatting of objects when switching between categories for example. That means that the client side knows how to format objects of type “product”. With a client/server-based architecture, the server would have to know how to format these objects and generates the source code, which the client browser would interpret at runtime for rendering. Figure 2 shows the differences in processing data between client/server based web applications and Rich Internet Applications.

Figure 2: Difference between client/server based web applications and Rich Internet Applications (top: client/server based, bottom: RIA)

Rich Internet Applications

The term “Rich Internet Application” [ALL02] stands for applications, which extend functionality and user experience compared to standard client/server-based web applications. This can be accomplished by adding custom user interface controls to the application such as navigation panes, accordions, date-selectors, etc. Also the integration of rich media such as audio and video can be seen as part of Rich Internet Applications. It’s difficult to narrow down the term RIA to specific characteristics but mostly they try to implement features the user is used to from desktop applications. RIAs can take over some processing of data to relieve the server’s payload but the main purpose is the client-side logic when presenting

data. The client itself is responsible how data will be displayed. In most RIA environments it's possible to sort or filter data on the client. In old client/server-based applications the sorting of data required a complete page refresh with the complete processing done by the server. That means that the client itself can do simple tasks. Therefore the term "rich" not only means rich media integration, it also means that there's a rich subset of functionality already on the client. Nowadays computers are not weak machines as they were used earlier when thin clients were used on networks and all of the processing power has been done on mainframes. Many client computers have more features and more powerful hardware than some Internet servers. And the client computer's performance is not used when just surfing through static (or generated) pages. But the server has to generate and calculate these pages for thousands of parallel requesting users. Rich Internet Applications use the processing power of the client computers to bring simple and not business-relevant functionality to the client to disburden the server.

The evolution of Rich Internet Applications

Rich Internet Applications follow a simple matrix of important facts for applications with the axes: "reach" and "rich" (see Figure 1).

The first axis "reach" stands for the population of applications.

The second axis "rich" stands for the kind interactivity for applications are the pure text based or do they allow interactive change of data.

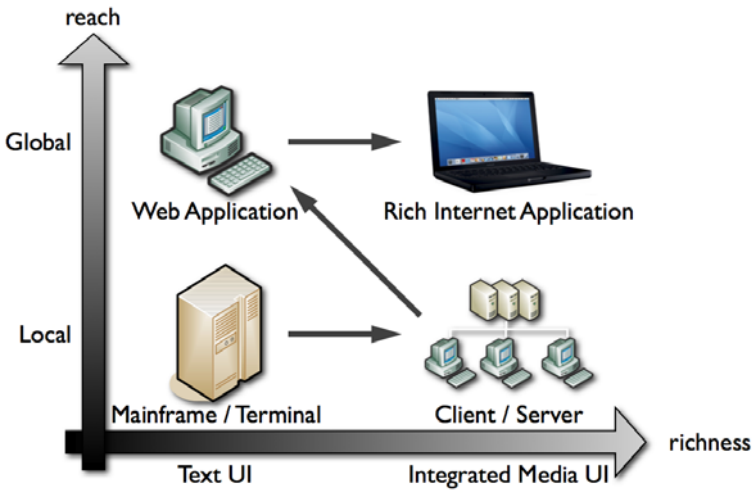

Figure 3: Evolution of Rich Internet Applications

In the early beginnings of computers and applications there was no “reach” for applications available. That means applications were installed locally on one machine. Due to the lack of networks there was nothing the application could communicate with. The interfaces of those applications were text-based and there was nearly no interactivity available within these applications. So these applications were not rich and had no reach.

Later the applications became more “rich”. That means with the beginning of graphical user interfaces (GUI) applications became more functionality and with pointer devices such as the mouse they became highly interactive. For example database applications allowed sorting or filtering of data. But those apps were desktop applications and were sometimes connected to a local network but not to a large network as the Internet. So these applications were “rich” but didn’t have “reach”.

With the beginning of the Internet era web applications became more and more popular. They followed the client/server principle and allowed the interaction with the largest network, the Internet. But the browsers were applications, which only could render markup language and very few scripts at the beginning. So the logic parts had to be done by the server. While those applications made a step forward to obtain a higher reach factor, they made a step back because of the lack of interactivity. These applications were not rich but they had reach.

The next and currently evolving step in application development is to write applications, which have reach and richness. These applications should look different when comparing them with their history? They should be able to connect to the network as web-based applications. But they should also be able to act intelligent in a way that not for all operations a server is needed. These applications then are rich and they have reach. Finally such applications are called “Rich Internet Applications”.

Note: A further step in the evolvement of applications is an “occasionally connected application”. Such an application would have both, reach and richness, because it originates from Rich Internet Applications. But these applications also need to have enough intelligence to allow the user to continue working when there’s no network connection available. Such applications have to keep data offline and save the data the user enters and synchronizes when there’s network connection available again.

The implementation part of this thesis is a Rich Internet Application built with Adobe Flex but as part of the requirement for the application it will also be deployed as an occasionally connected client as an Adobe AIR application.

Agile Software Development

When speaking about different software development processes, many engineers will think about large specifications of processes. In the past decades many different methods for software development have been widely used and are well know. The waterfall model, the V-model, the Rational Unified Process are just three of many different paradigms how to develop software projects and how to manage them. Most of these processes describe how to iterate different steps during development to improve the overall quality of the engineered product. They use quality standards, require domain-specific knowledge (for example for testing tools) and they require complete design specifications before the pure development part can start.

To show the difference between commonly used software development processes and the idea of agile software development two of the most-known development process will be described, followed by a detailed description of Agile Methods with some specific characteristics. The two already well-know processes described here are the waterfall model and the Rational Unified Process.

The Waterfall Model

The waterfall model has been very popular in software development for years. Mostly large software companies with large development projects to cope with use it. The waterfall model in its original version is non-iterative. [ROY70] That means that the engineers only can start with the next phase of the process, if the previous step is completely fulfilled. The waterfall model consists of five to seven main phases (depending on how to split phases), which are run through sequentially (see Figure 4).

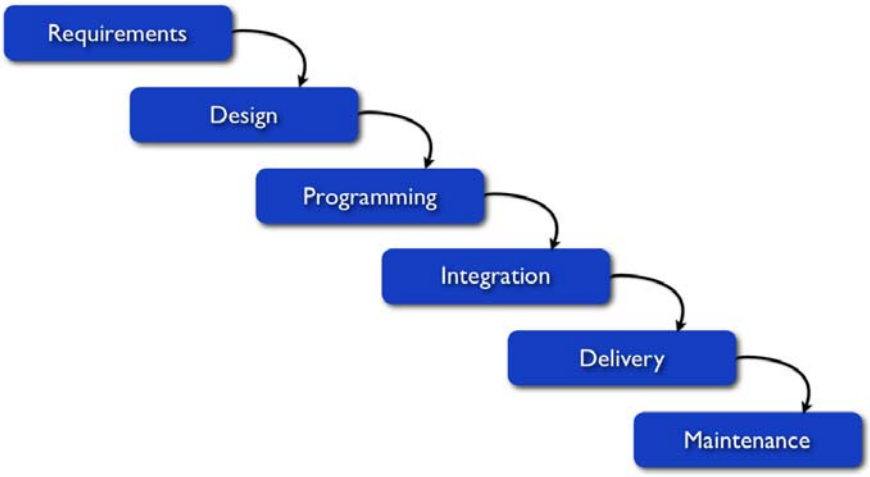

Figure 4: Phases of the waterfall model

The phases are:

1. **Requirements specification:**
   The requirements analysis contains the purpose of the application and describes how it should be used. This can be done by defining use-cases, functional and non-functional requirements, different constraints (design, performance, quality), interfaces to external applications … The Requirements specification also contains general information about the project and dependences to other projects.
2. **Design specification:**
   Within the design specification important parts of the system architecture are formally described. Depending of the kind of the application, different methods are applicable to describe the software process. This can be done by flow-chart diagrams, which describe the application process flow or this can be done by creating storyboards for user-interaction. As a formal method for the description of the application design the UML (Unified Modeling Language) is mainly used. UML is a standardized set of objects that represent different parts of an application and allow creating a graphical representation of the application architecture.
3. **Programming (sometimes Implementation or Construction):**
   Sometimes this step and the step before (Design specification) are

separated into three standalone steps: Specification, Design and Implementation. The main difference is that within the Design step there would be some basic algorithms already predefined and taken out from the Programming step.

The Programming step contains the development of the application. Based on the steps before there should be a detailed set of instructions, which need to be implemented by the programmers.

4. **Integration:**
   Within this step all developed modules are brought together and they are meant to work together. This step is called integration because different parts of the applications come together to build a homogeneous application.
5. **Delivery:**
   One of the main points of this step is acceptance testing by the customer to check, if the product meets the customer's expectations. If the product is accepted, it gets packaged and installed at the customer and finally will be used.
6. **Maintenance:**
   Even if the software meets the entire customer's expectations on delivery it's rather possible that over time new features are requested or bugs are found. To improve the software it enters the maintenance phase. Therefore it re-iterates all the previous steps, which results in a new version of the software on delivery.

One big drawback of the waterfall model is that the phases before implementation are taking much time. Often it's not useful to have detailed instructions for every small piece of the software. Sometimes it's better to find limitations and possibilities of the application requirements while developing and engineers can react on sudden problems. The waterfall model writes down the specification, which can't be changed (according to the process description).

The Rational Unified Process

The Rational Unified Process [EVE00] derives in its original state from the unified process. The main difference to the waterfall model is that it's an iterative

development. Basically the Rational Unified Process is use case driven (RUP is very familiar with UML), architecture-centric, iterative and incremental. Due to its closeness to UML, a basic principle of the RUP is to break down the requirements into functional requirements, which then are represented as use cases. During the architectural analysis also non-functional requirements are surveyed. These are mostly the common non-functional requirements, such as reusability, safety, maintainability, etc. The actual process of the RUP is iterative and incremental. Similar to Agile Methods, there are many iterations that result in an increment of the overall product development outcome. Four phases are defined within the RUP and represent the entire process. These phases can be split into several iterations with increments at the end of each of them (see Figure 5). [ÖHM05, KRU01]

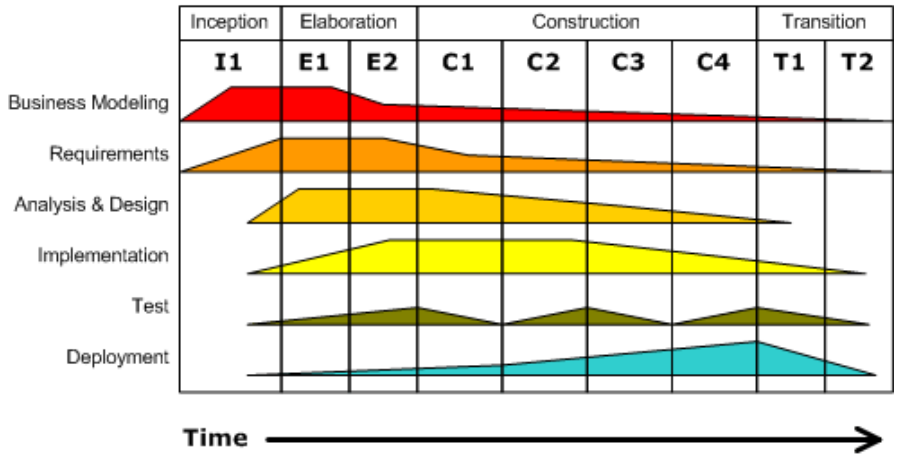

Figure 5: Rational Unified Process[1]

1. **Inception phase:**
   During the Inception Phase, the use cases for this iteration have to be defined and determined. All requirements have to be identified and the business case has to be developed. Also administrative tasks have to be done: who will work on the project and on which tasks, how much time is needed, a schedule has to be created. Risk management is also part of the inception phase.

[1] Image taken from: http://upload.wikimedia.org/wikipedia/en/0/05/Development-iterative.gif

2. **Elaboration phase:**
   The Elaboration Phase is used for the core architecture of the iteration. It's important to fixate the architecture for the upcoming construction phase, so that the construction can be done efficiently. Use case diagrams are created and the core components (from the architectural perspective) are created as classes. That means a small part of implementation starts during this phase, just to bring the architecture from the paper to source code. In the end of this phase, it should be clear that the planned architecture will work and finally a detailed schedule for the Construction Phase has to be planned.
3. **Construction Phase:**
   Based on the architectural definitions (and source code fragments), engineers and developers start coding the required functionality. The construction phase is usually the longest and largest phase within the project and should be split into several iterations. Iterations within this phase should be planned to represent single features or small feature groups to have a visible increment after the iteration.
4. **Transition Phase:**
   Within this last phase, deployment is done at the customer. This phase is also commonly split into several iterations to react on feedback of the customer.

Sometimes there are mentioned two additional phases: Production phase and Retirement phase. These phases are used within the Enterprise Unified Process and are out of scope of this short introduction to the Unified Process.

The Rational Unified Process is an improvement to the waterfall model, because it more iterative. Several iterations within some phases allow more detailed preparation for the upcoming phases. It's also important to start developing earlier than in the waterfall model, to get a feeling for the architecture and to check, if the planned architecture will work later on. Nevertheless the Rational Unified Process is also a heavy-weighted process with a lot of overhead at the beginning of a software development project. For web applications it won't fit completely because the customer identifies often functionality during the product development lifecycle and a lightweight process would be better to react on the customer's needs instantly.

Agile Software Development

The ideas of Agile Software Development were initially mentioned about a decade ago. Several factors lead to a rethinking about established processes.

One factor was that in the late 90s more and more companies got equipped with an IT infrastructure. Therefore custom applications were needed with domain-specific functionality. But these applications were not such huge applications than they were in the years before. And using heavyweight development processes for lightweight applications was not a clever idea.

Another cause was that more and more new programming languages and paradigms evolved that allowed a more abstract kind of programming. This permitted a less detailed architectural view and therefore not that much requirements and design engineering.

One of the most important motivations for a new paradigm was that developers didn't exactly follow models as the waterfall model or the Rational Unified Process. At least they could not work that effectively as they could without those paradigms.

Agile Software Development tries to bring the implementation part of the developer back to be the most challenging and most important part of the work. But it's very important to check the parts that were developed quite often to react on changes or find improvements immediately.

There are several challenges in product development that need to get solved. The following challenges lead to Agile Methods: [KRO07]

Note: the following text contains special roles and terms of the Agile Methods process. These terms will be explained later in detail.

Requirements are unknown, unclear, and not stable. Changing requirements or inserting new requirements is difficult.

This means that in most (or nearly all) "real world" projects it's quite impossible to determine exactly what the customer wants. Many features and functionalities are explored when people are working together on the project and not before the project has started. If there's a fixed specification at the beginning of a projects and the engineers follow a strict process (e.g. the waterfall model) it's not possible to change the requirements instantly or insert new requirements. The whole process has to be run through to create a new version afterwards.

Agile Methods however try to identify the needs and requirements of the product together with the customer. It's necessary to get the customer "on the boat" and have a commitment from the customer to use Agile Methods. Therefore the client has to name someone who can act as the "product owner" role during development and who is accessible for the engineers all the time. The product owner also is in contact with the "Scrum master" all the time and if the owner wants new features or has new requirements, he would tell it to the Scrum master.

Documentation takes up too much time (especially before implementation has started).

This comes as a result of the previous challenge: if one tries to determine all requirements that could evolve at any time for the product, this needs much time for specification.

In Agile Methods documentation is also important. Terms like "product backlog" or "sprint backlog" are the key elements of documentation before implementation starts. But opposite to classical software development processes it's unusual to write requirements specification sheets with hundreds of pages.

Project progress is not transparent.

When using conventional or traditional processes for software development, the project progress is only visible on the paper. The project leader may point out exactly, how much manpower has been used already and which features may have been implemented already. But until this moment, there are no releases of the software. That means it's impossible to have a look at a usable software prototype. Only when there are builds made it might be possible to have a look at them, but generally the project progress cannot be made visible by working releases of the software.

Agile Methods have time-boxed intervals for releasing running and stable software. The quality of the yet-developed features has to be at the highest possible level, as if this was the release for the customer for the rollout. Even if there are nearly no features developed, a working release is very important and therefore the project progress is transparent, it's easy to determine which features are already implemented and which are missing.

Integration is done towards project end and leads to unforeseeable problems and delays.

Late integration may lead to a “big bang” if in the end problems evolve which could have been easily solved at the beginning. Often application integration into other systems is a very complex process. Therefore it would be clever to check these things at the beginning to detect possible incompatibilities. At the beginning those problems often can be solved and the projects build on a solid base. If the interfaces to legacy systems have to be changed late, many workarounds have to be made, which leads to instability and a non-solid base for the entire product and the time for the workarounds is normally not planned and the budget for the project is exceeded.

Agile Methods do have regular and frequent releases. These releases are built to test their interaction with its interfaces to other parts of the environment. So it’s simple to determine possible challenges or blockers, which then could be handled soon to bring the product to a solid interface interaction.

Testing not executed properly due to a lack of time.

In software development processes like the waterfall model or similar other methods testing is often a fixed component in the process theory. But often the budget is very narrow calculated and if the steps before testing consume too much budget, testing is done on a very low level. When products have to be at the customer on time and there are some delays during development the last step has to be shortened. This may lead to big problems since errors found at the customer after release are usually more expensive as if they were found during development.

Developing with Agile Methods contains the testing part in each iteration and in fact on each day. Some specifications such as “test driven development” even have testing specified before development. (In TDD a test case follows the specification. After that the code has to be written that satisfies the test case. So the product assembles after solving all tests.)

Quality problems accumulate from release to release.

If testing is not executed regularly and properly, some errors might be undetected and software builds on a faulty base. If the base is faulty and there’s no time or budget to correct these errors, mostly workarounds are implemented. If this continues more and more such workarounds are building big parts of the product. It’s obvious that the unsolved errors may get bigger and bigger and at a certain point these errors must be solved. Then all workarounds, which rest upon, these errors (or underlying workarounds) need to be detected and solved. This may lead

to quality problems since it's not guaranteed that all these workarounds are found and corrected.

Agile Methods specify that each release, after each sprint has to be at a quality level of a production-ready product. Therefore the quality is more important than the number of features. If a certain part of the product is not production-quality, it won't get into the sprint-release. Only on acceptance it may be part of the release and errors won't lead into workarounds later.

Too many errors are detected too late.

As already mentioned above, late detected errors may result in big problems. But why are errors often detected too late in classical methods of software development? The answer is that limited time at the end of the development process causes limited time for testing. So many errors are not found or found too late (maybe at the customer).

Developing with Agile Methods assures that functionality only is inside a package, if it's well tested. And the tests are not done at the end of the product development, testing is done in each iteration during the development cycle (each "sprint").

Conflicts and problems are hidden and not solved.

Classical ways of development don't bother about personal problems or conflicts within the development teams. Agile Methods try to create a good environment for working and try to relieve the administrative burdens from the engineers. The "Scrum master" is responsible to keep problems and urgent customer requests away from the developers so that they can finish their work they just began without interruption.

These challenges show, in which direction Agile Methods point. They raise the problems of classical software development processes and try to improve these issues with Agile Methods.

As mentioned above, there are some domain-specific terms in use at Agile Methods, especially in Scrum, which are explained here:

Definition of terms in Agile Methods

Scrum is not just “hacking” and rapid prototyping. It also has different kinds of formalism that has to be followed. And it has some fixed time intervals with expected results that have to be accomplished. The illustration on page 43 (Figure 7) shows, how the process is working, where one can find the time intervals and which phases are part of Scrum.

Agile Methods are not based on a single big iteration over all steps during development such as requirements, specification, implementation and testing. Agile Methods iterate on all of these steps all the time, every month, every week, and every day. [HRU04]

- **Setup-Phase**:
  In the setup phase, several things have to be accomplished: the teams are built and the different roles are assigned to the personnel, working on the project. Also the basic requirements are specified and a certain document is created, the “Product Backlog”. The product backlog contains so-called user stories, which describe generally a To-Do list in a special manner.

  After the setup-phase has been done, the main part of the process is starting.

- **Sprint**:
  A sprint is a fixed time interval, where development happens. Typically sprints are 15-30 days long, depending on the size of the project and how many people are involved. During a sprint, part from the product backlog, the sprint backlog, is the list of tasks to do.
- **Daily Standup Meeting** (also: **Scrum Meeting**):
  Each (working) day, all team members and the Scrum master get together for a short meeting (max. 15 minutes). At these meetings, the Scrum master asks each person, what his goals for the past day was and if he could realize them. Also the Scrum master asks if there were any pitfalls or problems the team member ran across. If so, they are noted. Finally the team member is asked for his goals for the current day. All members are asked and the Scrum master summarizes the things he learned and updates the sprint backlog and the “Sprint Burndown Chart” according to accomplished tasks.

- **Product Backlog**:
  The product backlog is a table with entries like in a To-Do list. Each entry marks a “User story” and also contains a priority grade and a column for “story points”. The story points are similar to combined effort and complexity, the priority is set by the product owner and contains the priority, how fast this certain feature should be part of the product but on the other hand it also contains the risk for this feature. That means if there’s a high risk when implementing this feature because it’s very difficult or there’s only little experience with that specific domain and if this features is not possible to implement this would be the blocker, then this feature should get a high priority to get implemented first. Before implementation is started then, the product backlog is ordered by priority, so that the high-prioritized items are on top. The team and the Scrum master then decide, how many user stories are part of the next iteration, the next sprint and these user stories are then transferred to the sprint backlog.
- **Sprint Backlog**: The sprint backlog contains the user stories for a single sprint. Within the team, each developer can choose, for which user stories he’s responsible. Each user story is normally broken down into different tasks. Each of the tasks should be done within about one day, so that at the next daily meeting it can be marked as done or not done. The tasks should possibly defined very small and understandable, if a tasks is so big that it will take much longer than one day, it should be split into smaller parts.
  The sprint backlog should be finished and all user stories marked as done at the end of each sprint.
- **Sprint Burndown Chart**:
  The sprint burndown chart (an example can be seen in Figure 6) is a 2-axis diagram that reflects the progress during the sprint. Based on the progress the team and the Scrum master may decide to remove some user stories from the current sprint and postpone them and put them back to the product backlog, if the chart shows that time is running out for too much user stories to be done. The x-axis shows the remaining time in the sprint whereas the y-axis shows the remaining effort to do (planned hours, tasks, user stories, etc.). In a sprint burndown chart there are always two lines visible: one line, which shows what the progress should look like if everything goes fine and there are no delays and the second line shows the actual project progress. If the second line is above the straight line, the progress is going to slow to meet the expectations in the end. If the

line keeps staying above, the Scrum master has to discuss with the team, if and which user stories may be postponed to the next sprint. If the line is below the planned line, the team works faster than expected.

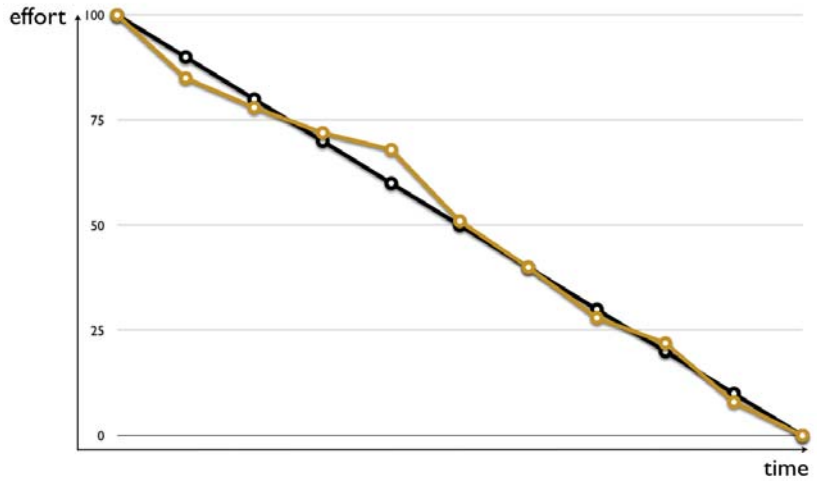

Figure 6: Sprint Burndown chart

- **User Story**:
  A user story is an entry in the product backlog that describes certain functionalities for the product. A user story should be written down in the following format:
  As a <role>, I want to <action> so I can <benefit>.
  These sentences should be easy to understand and make clear, what has to be done.
  Examples for user stories may be:
  - As a program, I want to have an API method for server interaction so I can use the server for a calculation.
  - As a user, I want to have a dialogue, where I can input my address data and contact details.
  - As a user, I want to search for my colleague's name, so I can find his telephone number and call him.

  Based on such user stories it may be great, if there will be a design mockup for each user story. It's helpful if these user stories (plus the mockup) are drawn on small paper cards. Writing user stories on paper cards has three advantages: First,

they don’t get too long. A user story should be written short and precise, so that anybody understands its purpose. Second, when putting theses user stories on a pin board, one can keep track of its progress and who’s assigned to that user story. And third, with the simple mockup, there’s already the first idea, how to design the software. With many user story cards and mockups, one has an overview over the entire application.

Scrum uses different roles for the people working for a project. Based on the theory, there are at least three roles, which are part of the personnel. [SZA07]

- **Product Owner**:
  In a perfect environment the product owner is an employee by the customer. He’s responsible for all details for the product in question. He represents the customer, his requirements and wishes. The product owner also assigns the priority of features, which should be developed in the product backlog. If this person is not the customer itself, this person should be as informed as the customer in any perspective of content and be accessible at any time for content and requirements-related questions for the team and the Scrum master. During the sprint the product owner must resist to change items in the sprint backlog because when the sprint is in progress things must not be changed due to the regulations of agile development.
- **Scrum Master**:
  The Scrum master is the person, which is responsible for the team. He acts as a shield between the team and the product owner. He ensures the productivity of the team and tries to avoid interruptions in the workflow of the team members. This person is also responsible that the Scrum process is done right and he’s the moderator of the daily meetings. The Scrum master ensures that the sprints are finished on time and that each increment (after each sprint) is a potentially shippable version of the product. (Potentially shippable just means that the delivered parts are working as expected and does not mean that any functionality is in the release but don’t work correctly.)
- **Team**:
  The team consists ideally of seven (± 2) persons. The team should work in a cross-functional way that means that there are programmers, designers, testers, etc. in the team so that the team can manage itself and doesn’t require interaction with other teams during a sprint. Often it’s useful to have the team together

in one team room during product development. The team itself has freedom how it accomplishes the tasks that need to be done. They can figure our by themselves how to turn the user stories from the sprint backlog into working functionality after the sprint.

The Scrum process

When talking about agile development, one might think that “agile” may mean chaotic. Only because a team works in chaos and that works, and the results are good, this might be agile because they react fast on changes and so on. But Agile Methods do have very strict rules to follow. Scrum, as a special model of Agile Methods, which is commonly used when talking about Agile Methods. The most known model in agile development is XP (eXtreme Programming), but it’s also the most extreme example of alternative development strategies. When using Scrum the first time, it might be difficult to set it up for all of the involved persons: it’s hard to define all needed specifications and to find the right people for the different roles. The Scrum process itself is very strict to follow. One important keyword is “time boxed”. In Scrum, nearly everything is time boxed. That means, every meeting, sprint iteration and daily Scrum meeting has a fixed length and should not be shorter or longer. The following illustration shows a (perfect) flow for Scrum: [HRU04, KRO07]

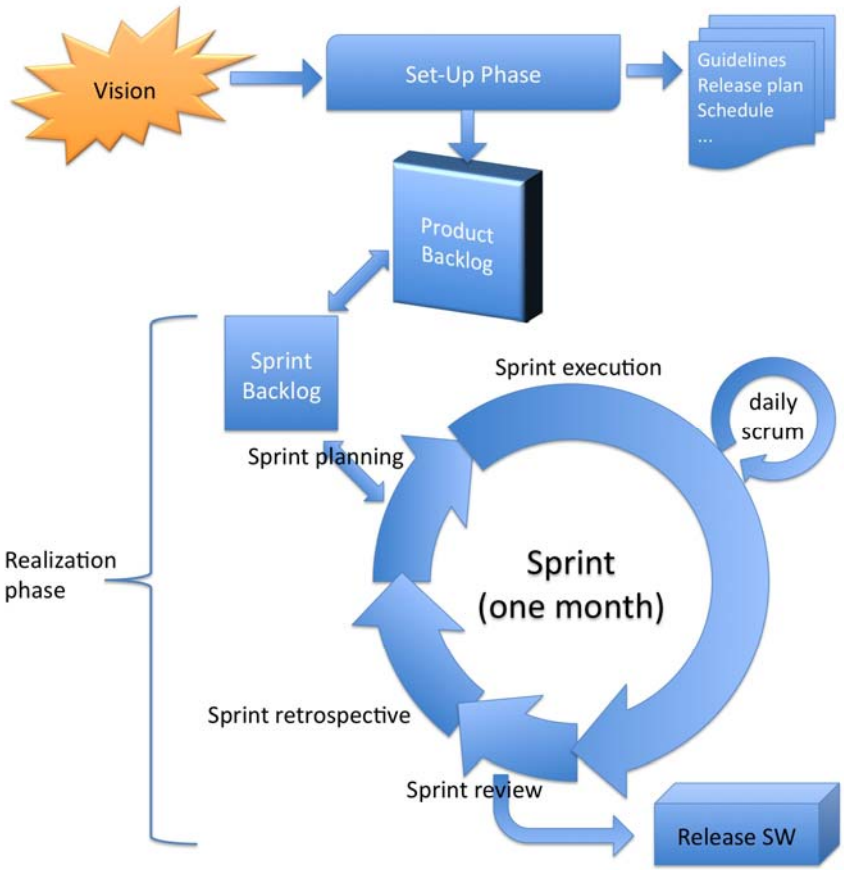

Figure 7: Scrum Flow

- **Vision**:
  At first, when initiating a project with Scrum, there's the vision. The vision might be an idea for an internal project to develop, or a won tender for a customer and the order for an application.
- **Start up phase**:
  In the start up phase the teams are defined that work on the project and how they manage themselves. Also an initial product backlog will be defined and adjusted with the product owner. A release plan is defined and estimation is made on how long the implementation will last. Also administrative decisions and documents are created, such as coding guidelines, etc.

- **Product Backlog**:
  The requirements have to be defined and written down into the product backlog. But that's not a typical requirements engineering, instead the team works with the product owner, to define all functionalities, which should be part of the application and also other tasks, which are part of the project should be written down into the product backlog.
- **Realization phase**:
  Once the product backlog is finished, the realization phase begins. This phase is called sprint and lasts, under normal circumstances, one month. As described above, a sprint should be seen as atomic and should not be interrupted by change requests by the product owner.
- **Sprint Planning**:
  The first part of the sprint is the sprint planning. Sprint planning is normally a meeting between team, Scrum master and product owner. The Scrum master is responsible to invite all necessary people to the meeting and he also acts as a moderator. Together with the team, the product owner creates a sprint backlog out of the product backlog and the sprint backlog is seen as the sprint goal. Once the sprint backlog is complete, the sprint execution phase can be started.
- **Sprint Execution:**
  In this phase, the team is responsible for the implementation of all needed functionalities defined in the current backlog. Not only implementation is important, also testing, designing, etc. is done during this development step.
- **Daily Scrum Meeting:**
  Each (working) day of the sprint, the Scrum master initiates the daily Scrum meetings. There the Scrum master and all team members get together for a short meeting. Often the name "daily standup meeting" is used because these meetings should be done standing and very short. The Scrum master then asks every team member about his progress with three questions:

  – What have you been doing during the last Scrum period (the last day)?
  – Were there any problems you ran across and what those may affect?
  – What's your plan for the next Scrum period (this day), what will be your tasks?

  Based on the answers of these questions, the Scrum master updates the sprint burndown chart to reflect the progress of the sprint.

Another job of the Scrum master is, to act as a shield between the team and the product owner. The team should be able to work without disturbance from the customer. If there's really a big request for interruption, the Scrum master has to decide, if and how the current sprint will be interrupted.

- **Sprint Review:**
  The sprint review is a meeting at the end of the sprint. At this meeting again the team, the product owner and the Scrum master are involved. The team presents the accomplished work to the product owner. Based on the satisfaction of the customer, some items on the sprint backlog may be approved and some may be pushed again into the next sprint backlog for improvement tasks. At this point the product owner may decide, if the achieved result is released on to the production system or not. The quality of the product at the time of a sprint review should always be production-ready, so that the customer may use the product instantly.
- **Sprint Retrospective:**
  During the sprint retrospective meeting, the team and the Scrum master talk about problems and chances of the past sprint. Together improvements are discussed and how they can be implemented during the next sprint. The Scrum master also checks the agility status of the project.
  After the retrospective, the sprint planning starts again (if there are user stories left on the product backlog).

Agile development for web-based applications

Note: This chapter is based on the own experience of the author when developing web applications using Agile Methods.

Web-based applications are mostly a set of functionalities bound together into a portal solution. Many customers need applications for intranet solutions, e.g. travel management, WIKIs, forums, content management systems, etc. Also extranet applications are needed, to stay in contact with customers, e.g. for booking systems, claim management, contact, etc. These applications are often only a few components, which need to be integrated to work together. Therefore a big part in developing web-based applications is integration. Integration is difficult to be split

for Agile Methods. The rest (writing components for functionality) is easy to split. As an example a CRM (customer relationship management) application is used (only a small part, a complete CRM application would be too large to describe here). The CRM system should have the following features:

- Add / Edit / Delete contacts
- Add / Edit / Delete companies
- Assign contacts to companies
- Manage orders and assign them to contacts / companies

Based on these requirements, the product backlog should be created. As described above, the backlog consists of several user stories, which describe interaction with the application and contain priority and effort estimation. The effort may be a real number for hours to work on, or just a descriptor that indicates the size of the task (such a description could also be: XS to XL, as these "numbers" are used for T-shirt sizes). The product owner and the team then together are setting the priority, respecting both, technical priority and content priority. An example Product Backlog can be seen in Table 1.

Priority	**User story**	**Effort**
1	As the server admin, I want to set up the test server, so the developers can start implementing the application.	8
4	As the designer, I want to create designs for the application, so the developers can concentrate on implementing features.	40
2	As the user, I want to manage contacts, so I can use them later for relationship management.	20
2	As the user, I want to manage companies, so I can use them later for relationship management.	20
3	As the user, I want to create relationships between contacts and companies, so I can see who's involved in which company.	10
3	As the user, I want to manage orders from customers, so I can see a history of the work from my company to the selected company.	24
3	As the user, I want to have an overview of all projects I've ever done, so I can see if there may be a customer who may	24

Priority	User story	Effort
	need another product from me, when it's missing in his job list.	
1	As the DB admin, I want to create a database schema, so that developers can leverage on that and it will be extensible for future purposes.	12

Table 1: Example for a Product Backlog

During the sprint planning the next step is to take some user stories out of the product backlog and put it into the sprint backlog.

DONE	**User story /** Task	Effort
	As the user, I want to manage contacts, so I can use them later for relationship management.	**20**
	Create a view class based on the designer's guidelines for displaying a list of contacts.	2
	Create a view class for contact manipulation (form).	2
	Create structural classes for events, commands and delegates.	3
	Create value object class with properties to match the fields defined in the database.	1
	Add validation classes to required fields	2
	Implement handlers for clicking.	2
	Implement methods for create / edit / delete	4
	Set up formatting classes to display special properties in a nice way (e.g. Date formatting)	2
	Testing	2

Table 2: Example for a Sprint Backlog (one user story and corresponding tasks)

In Table 2 a new column appeared, called "DONE". This column should be checked, if a certain task has been done. The term "DONE" should be defined during the set-up phase. It has to be clarified, if e.g. testing is part of the task or should be mentioned separately, or if the Scrum meeting with the drawing into the burndown chart is part of the tasks / the user story.

Looking at this table it's obvious that one single -developer should implement this user story. The tasks are very small and it wouldn't make sense to split these

tasks to multiple persons. Therefore it's usual that developers have several user stories at once to work on during a sprint. Even if sprints are shortened to one week, it's possible to have several tasks, depending on the granularity of their description in the product and sprint backlog.

During implementation of these tasks, there are daily Scrum meetings. The developer will be asked about his progress and the progress will be updated into the Scrum burndown chart. At the end of the sprint this feature should be implemented completely. If not, its release will be postponed to the end of the next sprint. If the feature is ready it will be packaged during the sprint retrospective phase and delivered (if applicable) to the customer.

During the sprint review and sprint retrospective phases, developers and the Scrum master discuss if there were any problems and how these problems could be solved in the next sprint. When this discussion is over, the next sprint begins with the sprint planning.

Agile Methods for very small projects

Normally Agile Methods scale very well in project with different sizes. Due to the possibility of hierarchies for teams, even for very big projects Agile Methods may work [HRU04]. But for (very) small projects, Agile Methods don't work that good.

Note: This part of the thesis is based on the experience of the author during the development of the practical part for this topic. The fact that Agile Methods don't fit in any project of any size does not mean that there are no parts of the process, which can be used. In fact, there are some ideas of Agile Methods, which should be used in all development projects.

There are several factors, why the original process and rules for agile development won't work for very small projects. An example will describe, where the blockers may be and at which sizes of projects Agile Methods won't fit in its entire form.

As an example, if a team consists of five employees (the lowest number, mentioned in the recommendation for Agile Methods) and a sprint that will last one month there may be a problem. If there should be agility in a project, sprint meetings are essential, therefore at least two or three sprints should be performed until

the project is finished. Assuming that a sprint lasts one month, three sprints last three months, such a project with five team members consumes 15 person months. That's not a small project. Small projects are usually projects between two and eight person-months.

Even if fewer people work on a project, let's say three people, and the sprint is shortened to be bi-weekly, for a normally two-person-months project there's just one sprint. That's because three people × 0.5 months = 1.5 months. This means, there is no more sprint available after the first one.

And for the "big", eight-person-months project there are five sprints. This is because five three × 0.5 months = 1.5 months and 1.5 months × 5 would be 7.5 months, which is close to the targeted eight-person-months project.

Vice versa that means if a team consists of three people and the sprint is shortened to two weeks and there should be at least five sprints, the minimum time for the project would be eight-person-months. And that's only pure working (development) on the project. There's no overhead for Scrum and sprint meetings, and the product owner and Scrum master are also not accounted yet. Estimated that these two roles and the Scrum and sprint meetings are accounted by ~20% of the overall time (called "overhead"), that means the smallest project has to be nine person months for this slimmed-down version of Agile Methods. Normal iterations (monthly) for sprints would cause that the minimum is 18 months. And with five members this will go far over 30 person months, which are more than two and a half person years.

Table 3 will show typical numbers for agile development in small projects.

# Persons	Sprint length (months)	# Sprints	Development time (months)	20% overhead	Total (months)
3	0.5	3	4.5	1	**5.5**
3	1	3	9	2	**11**
3	0.5	5	7.5	1.5	**9**
3	1	5	15	3	**18**
5	0.5	3	7.5	1.5	**9**
5	1	3	15	3	**18**
5	0.5	5	12.5	2.5	**15**
5	1	5	25	5	**30**

Table 3: Examples for agile development durations in small projects

When looking at this table the shortest project with three people using agile development methods “light” would be nearly six person months. And the fact that small projects are commonly projects between two and eight person-months, six months would only represent the upper third of these projects. The shortest possible project with “standard” agile parameters (five persons, sprint length is one month) would be 30 person-months.

Given the fact that Agile Methods in general are commonly understood as quite efficient and effective mindset as a software development process, this calculation shows the method’s limitations.

But besides the fact that not all Agile Methods ideas are suitable for very small projects, a number of concepts should be used anyway:

- **Agile procedure with the customer:** When the customer is willing to use concepts of Agile Methods, it would be a good idea to do documentation and meetings together with the customer to save time for development. It’s always a good idea to have a close relationship with the customer because a steady contact may prevent misunderstandings and thrives business connections.
- **User stories and tasks:** In small projects there’s often very little documentation. Sometimes there’s a requirement specification or a short proposal document. When there’s an agreement with the customer it would be better to write detailed user stories and tasks instead of such documents. Based on these user stories and tasks, it easier to create documentation (if needed) and it also helps the understanding of the progress during development.
- **Sprint planning and review:** Even if the customer is not available at all times for the developers / the team, a short sprint planning and a sprint review is a good idea to give all involved persons a status update over the progress of the implementation. These meetings don’t have to be on a strict regular basis but they should be done for informational purpose. For all developers it’s important and interesting, which features are already done and what the colleagues are working on. Detailed analysis of the progress by drawing sprint or Scrum burndown charts is not necessary, within small teams and short periods it’s enough to speak about finished user stories or tasks. Short demonstrations of new functionalities may also help to understand the progress and is mostly exiting for the colleagues.

Finally, Agile Methods don't work in their entire process scope for very small projects, but it's important to pick out several ideas of Agile Methods and use them even in such projects. Not only methods, which are defined in the specialized version of Agile Methods Scrum may be applicable, also methods like eXtreme Programming or Pair Programming may be a good idea to try.

Agile Methods for single-person projects

Agile Methods for single-person projects are a continuation of the theory of Agile Methods for very small projects. Usually at Agile Methods the "Team" is the center of the entire software development process. Without a team and its communication, it's difficult to act "agile". But sometimes projects are so small, that only one single person is going to work on it.

Is it possible to embed Agile Methods in such a diminutive development process?

Agile Methods in its entire characteristics won't fit for these kinds of projects, but again, as described above, some ideas can be used and should be recycled. Most of the time developers familiar with Agile Methods use such ideas implicitly, without thinking about the origin of their doing. Single-Person projects commonly don't use large and overweighed software development processes like the waterfall model and developers also don't write long requirement specification documents when developing alone. Their approach is mostly to define certain To-Do lists, with features to be developed. If someone really wants to act as agile as possible, this list can be written using the pattern of User Stories. To calculate the overall effort for the project and present estimation for the overall project costs to the customer, these User Stories are evaluated in their complexity and finally a development expense is defined. When using the agile approach, it would make sense to make a note on the implementation complexity and the "blocking factor", which means the problem, which may exaggerate, when developing this certain functionality.

When all of these things are done, the developer has created a "Product Backlog light", a list with User-Story-like To-Dos, and their estimations on complexity and effort.

Many developers are using a time-management system to keep track on their development time. The author's system for time management during work is called "TimeCards" and allows a detailed description of done work.

To understand the further intention, why such time management tools may be important for Agile Methods in single-person projects (and may even be more important in larger projects), the TimeCard system has to be described shortly for better understanding:

Time cards are single work tasks, which can be entered and described within a web-application (described in [DIM05]) for weekly and monthly reports. A single time card consists of:

- Type of work
- Date
- Time (from and to)
- Project
- Task
- Technology
- Description

The type of work is chosen from a list, which contains entries like: Meeting, Programming, Content, Know-How, Administrative, etc. The Project is a list of projects, the developer works on. Task allows entering small packages of the projects the time card can be assigned to. Technology lists a selection of technologies, which the developer uses. Description is a free text entry box into which the developer describes what he did within the timeframe of the time card.

Time cards are normally small tasks between one and four hours. That means usually 3 to 5 time cards a day are common, and mostly these time cards have different tasks assigned, depending on the flexibility and responsibility of the developer.

The TimeCards system also allows the dynamic adding of new projects, tasks and technologies, therefore there's no administrative need for project setup, etc. Each developer may have his own tasks or can share the same tasks with other developers. Finally, TimeCards allow detailed reporting on the developer's work. Reports of used technologies are available as well as detailed summarized information for projects and their sub-tasks, and how much time has been spent on certain functionality.

Using a time management system like TimeCards or something similar may help applying agile methodology to software development projects. Having a detailed report for each task (which should be mapped to tasks within User Stories), a project and event sprint burndown calculation can be done.

In summary, Agile Methods don’t work perfectly for single-person software development projects. But with a good environment and planning, developers can reuse ideas from Agile Methods when working alone, and profit from this basic and very little administrative overhead, when working in large agile-driven development projects.

Usability in web-based applications

Usability in web-based applications has to be treated slightly different than usability in classical desktop applications. In desktop applications there are mostly user interface guidelines, which describe, how applications should look like. One of the most-known and detailed guidelines is from Apple Inc., which describe in their “Human Interface Guidelines” how desktop application on their operating system, Mac OS X, should look like [APP08]. Such guidelines for user interfaces also contain large parts about usability. Positioning of buttons is not only a design decision, it’s also very important for usability. Users expect certain positions for buttons and menus. At least they expect a consistent look and feel throughout applications of the same manufacturer or on the same operating system.

Other things described in such guidelines are the behavior of applications on certain user interactions. These guidelines describe what should happen, when a user clicks a close button; or what should happen, when a user drags and drops objects or when a user moves objects to the trash. Also detailed descriptions are made for interactions with spreadsheets, double-click handling, application icons, menus, etc.

Most of these specifications are only suitable for desktop-applications. The majority of web-based applications don’t use spreadsheets or don’t even user menus. That’s because browsers don’t support such features by default. But in the last few months and years a new term evolved, named “Web 2.0” and a new era for web-based applications started. Technologies like AJAX and Flash allowed new user interface elements, which could be used easily in web-applications. New forms of navigation elements like accordions, menu bars, stacks, etc. came with these technologies, which were not parts of typical web applications before. Therefore it’s important to have guidelines, how to use them in an appropriate way, so that the user is not confused when using them and knows, how to use them correctly.

But usability is not only the term, which describes the interaction of the user with interface controls (see page 57 for a detailed list, what the common understanding for usability is). Usability is commonly known as the degree of ease, how

a user can use a tool to achieve a certain goal. Therefore important parts of usability are also efficiency, effectiveness and satisfaction.

For engineers and designers it's important to think, who will use the applications they develop. Based on these facts, the user interface and the application behavior should match the user's skills and needs. It's also a need that the context of the application has to be considered. If the application is a standalone application (even on the web), there are fewer restrictions on how the applications should look like or how its look and feel should be. But if the application is part of a bigger set of applications, it must fulfill eventually defined user interface guidelines from the parent application. For example if there's a company intranet and there's a web-based travel management tool, which should be integrated into the intranet, it would be extremely helpful for users, if this application meets the expectation to be consistent in its usage and look and feel with other intranet applications. If that's not possible, e.g. because the applications is an acquired application with no modification options, it should be launched externally and not in the context of the parent application.

What's also extremely relevant in terms of usability is efficiency. It's important to measure, how much time a user needs to do certain tasks. And it's also important, how much time the user needs doing the task the first time, without previous knowledge and help, and how long he needs, when doing the task very often (e.g. because it's a recurring task which has to be done again and again). These factors are very important for the measurement of usability. If the user needs very long time, even the task is recurring, there's room for improvement in usability. To cope up with problems that first-time users behave different than frequent users, e.g. users can be enabled to use shortcuts for things, they use very often [SHN].

But there are some other important facts in terms of usability for web applications, which are not technology related. It's important to test, how users behave on the website, which parts are interesting and what the user is looking for and what he can't find.

There are different approaches to test these factors. On one hand, before developing a website it's important to ask real users, what they expect from the site. For example, for an Intranet web page the employees of the department are the key users, therefore they should be asked, what they expect to find there. In terms of acceptance, it's very important to let employees be participating during software planning and development. [PUS08]

On the other hand, sometimes it's not possible to ask users before launching a website, what they are expecting. This may be the case, if a new online shopping website should be launched. Then it's important to start surveying users, as soon as they start using it. For example during the checkout process, there may be a checkbox, questioning: "Would you like to use PayPal as payment method in the future?"

Finally usability tests can be performed to detect the problems of a website. Instead of large usability tests, also a small group of individuals should detect the major problems on a website. According to Jakob Nielsen, it's enough to perform usability tests with 5 users. Five users find 85% of all usability problems [NIE00]. If there are problems detected, alternatives for the current solution for a task should be developed and tested again with the users. The test persons should identify the best solution for the overall system and in terms of usability. It's also very important for the developer and the customer (of the website, not the test person) to watch the test persons working. So the may get insights, how possible customers (of the products from the website) work on the website, how they use it and where to improve workflows for the users to deliver him a better experience and bring him back for another purchase.

Design principles for better usability

There are many different metrics to measure usability, e.g. with eye-tracking technologies or the length of the way of the mouse pointer. But to describe how to improve usability it's difficult to define exact rules. Due to the fact that usability can't be objective per se, there only can be recommendations or guidelines how to evaluate and then improve usability.

One of the pioneers of usability topics is Jacob Nielsen. In different papers he describes heuristics for user interface design. One of the most known publications is the list of "Ten Usability Heuristics", which contains ten so-called rules of thumb [NIE05] from his book "Usability Engineering" [NIE93]. A detailed description of the meaning of these rules can be found on page 62 where specifics within Siemens are mentioned.

1. Visibility of System status
2. Match between system and real world

3. User control and freedom
4. Consistency and standards
5. Error prevention
6. Recognition rather than recall
7. Flexibility and efficiency of use
8. Aesthetic and minimalist design
9. Help users recognize, diagnose, and recover from errors
10. Help and documentation

Nielsen developed these principles back in 1990, when the World Wide Web didn't exist at all and shortly after the first web browsers were created those weren't capable of technologies as nowadays. Although the most principles can and should be used today as well there have to be some corrections or enhancements to these principles. For standard web applications, which are HTML based, some paradigms may not work, for modern Rich Internet Applications, which tend to imitate desktop applications these rules may start to apply again.

To have a better look on these guidelines for HTML based web applications and Rich Internet Applications, a detailed list of pros and cons will be created:

1. Visibility of System status

   – HTML based applications are commonly not able to display the system status always up-to-date. Normally, such status updates can only be delivered to the user on page requests. If the system is down, a page request cannot be done and the application interaction has to be ended. (Although some browsers allow browsing back in history, the application ended without prior notification).
     Not only the system's up- or down-status should be displayed, also during interaction (e.g. uploading data, etc.). HTML does not have appropriate tools for displaying such interaction, although there may be some tricks with JavaScript, but these methods are not common ways to inform the user what's going on.
   – Rich Internet Application may be able to display the system's status and inform the user that the host may be currently unavailable. This can be done via push notification from the server, when there's a planned maintenance timeframe, or on data submit via services that leads to errors. However the Rich Internet Application has not to be quit at that moment be-

cause it’s running on the client and no page refreshes are necessary. The user may wait until the host is back online.
Rich Internet Applications can display what’s going on, during server interaction (e.g. when transferring data via Web Services, etc.). It’s also possible to show progress information when uploading files, etc.

2. Match between system and real world
   - Both, HTML based web applications and Rich Internet Applications should be able to speak the user’s language. The main idea for this guideline is that the user should find ways of interaction natural and descriptors are commonly understandable. There are no limitations for HTML applications or RIAs that may prevent these improvements.
3. User control and freedom
   - Nielson’s intentions with this idea were, to provide functionality such as Undo. HTML based applications don’t support such functionality. Only very basic methods, like Forms reset, are available. The browser’s history functionality may help in certain environments.
   - RIAs also don’t support Undo and Redo out-of-the-box. But some frameworks, such as Adobe’s Flex do have APIs for such functionality. It’s also possible to use the browser’s history (Back and Forward buttons) to switch between states.
4. Consistency and standards
   - HTML based web applications mostly use standard user interface controls such as buttons, text inputs, etc. Although these controls can be styled via style sheets, users generally recognize them as standard controls. But there are some drawbacks because HTML does not support all user interface elements the user may know from client applications, such as tabs, menus, etc. HTML navigation for example, works different than the File / Edit / View / … menus in desktop applications.
     In general users know that they are working with a web application and they have a concrete knowledge how to use them. Often web designers try to escape the rules for web applications and design their web applications to look like desktop applications but this is not the way to go. People using

a web application want to use a web application and not a hybrid between bad application designs and web design principles.

- Rich Internet Applications also use standard controls, the users are aware of. Most of the time, these controls don't use the same look as those coming from the operating system's default settings. On one hand they don't look that different than standard controls, on the other hand, they can be styled to look like the designer wants them. Rich Internet Application also introduce new controls, the user didn't know earlier, e.g. Accordion controls. These controls allow saving space on the screen and group similar inputs together and providing a better experience for the user. For example the most common inputs can be placed on the first accordion pane, more advanced inputs, which are not necessary for all cases can be placed into the "advanced" pane. Such individual or complex controls have been introduced by Rich Internet Application and are now ported back to standard controls; even in HTML based applications JavaScript libraries allow such complex user interface elements.

5. Error prevention

   - HTML based applications are able to check for errors in user inputs via JavaScript. Either one can develop such algorithms by self, or there are lots of libraries available for download, which help with such common problems.
   - Rich Internet Applications also do have error prevention and forms validation bundled. They even allow checking for errors during user input and marking visually, where the error occurred.

6. Recognition rather than recall

   - To enter data into forms, basic HTML applications don't offer many different user interface controls. Plain text input fields, text areas, or drop down menus are available. It's up to the developer, if he implements inputs as drop down menus (if applicable) or to let the user input the data manually. If a choice is possible, it's always better to let the user choose instead of letting him input data manually. This could lead to typing errors or even semantic wrong values, if the user didn't understand the purpose of the

field correctly. Normally there's no limitation in standard controls with HTML based applications.

- Rich Internet Applications do have the same basic input fields such as HTML forms do, such as text inputs, text areas, or drop down menus. However there are some additional controls too, which help the user to reduce his memory load, e.g. there are pre-defined actions for drag and drop, that means a user may drag items to their places instead of typing in values into forms. Although this may sound handy, it's also important to keep the alternative available, because some power-users may prefer to enter values manually instead of using the mouse for drag and drop.

7. Flexibility and efficiency of use

   - HTML applications don't support accelerators generally. Because HTML applications are normally just for data display and data manipulation and following data transmission, object manipulation for acceleration is not available.
   - Rich Internet Applications understand and use a rich data model. This leads to better object manipulation and full object control. That means, Rich Internet Application may, e.g. clone objects. Such functionalities allow the user to change only small parts of the objects, and save it, instead of creating the object as a completely new object and add all values, which are already input in another object just to change a single property. Such accelerators allow the user to act really fast and help him to reduce the time of data input dramatically. Rich Internet Applications also allow partial objects, e.g. when the user starts to fill a form, he can navigate within the RIA and come back and the form still is the same with the already input data. HTML apps wouldn't support that, except the server keeps all input data at all states, which would be very difficult to implement.

8. Aesthetic and minimalist design

   - Designers can design HTML apps and Rich Internet Applications individually. There's no limitation, not to design them aesthetically or minimalistic.

9. Help users recognize, diagnose, and recover from errors

- HTML applications and Rich Internet Applications both support instant notification on user errors, such as syntactic wrong user inputs, etc. It's up to the developer to create error messages, the user can understand and they don't confuse the user.

10.Help and documentation

- HTML applications may contain documentation just as RIAs. Rich Internet Applications allow searching within the documentation due to its rich data capabilities and instant filtering methods. It's also possible to embed audio or video into the documentation, generally for both types of web application, HTML based and RIA.

Usability at Siemens

Within Siemens there are also usability criteria for application development. A dedicated Support Center "Usability" created the checklist based on the ISO standard 9241 ("Ergonomics of Human System Interaction") and also adapted Nielsen's usability heuristics. These Siemens-internal guidelines were created especially for the Siemens Engineering Methods (SEM), which are the base for all software projects [SEM] and are manifested in a ten-topic-structure:

1. **Simple and natural structure of windows / dialogue sequences:**
   In general, dialogues shall only contain basic information, so that the user won't get distracted by unnecessary information. Often it's better to have a multi-step assistant dialogue than a huge dialogue with all input fields in one window. Similar information should be grouped together to help the user to work as efficient as possible when he doesn't have to switch between different topics.
2. **Visibility:**
   It's important for the user to understand, what he's working on. It should be possible to identify, which window is currently active and within which form field the cursor resides. In terms of visibility it's also important that the user is able to view the whole content of an application window without the need to scroll.

3. **Speak the user's language:**
   While developing software, engineers have to thing about the typical user of the application. End users often don't understand terminology that's vocabulary of developers. The user should not be bothered with technical details, why some error occurred, instead he should be given advices how to prevent such errors. It's not always important to have localized versions of software. Even if there are localized versions available, sometimes it's better to keep the original (mostly English) term, if this term is well known as foreign word and even used in the natural language of the user.
4. **Minimize user memory load:**
   By memory load the need to think about unnecessary things meant. The user should not be forced to keep everything in his head during software interaction. That means that the system should help the user when filling out forms by e.g. displaying context-relevant help or input tips. It's also better to have a list of available items than forcing the user to keep all items in his head and let him fill out the form with memorized values.
5. **Consistency and compliance with standards:**
   On one hand consistency means the same look and feel of dialogues and interface elements throughout the entire application. On the other hand, the application should also respect user interface guidelines from the operation system layer above. Buttons should look similar to buttons that are used in standard operation system dialogues. Web applications don't have to use the exact button layouts that are used, when displaying non-styled HTML interface elements. Users are used to that HTML based applications are styled and buttons don't look the same always. But desktop applications should at least try to keep a consistent design with the operation system's design. Therefore some vendors of operating systems created detailed guidelines, how to create applications that integrate well into the operating system (e.g. Apple Inc's "Human Interface Guidelines" [APP08]).
6. **Feedback and good error messages:**
   The system should always let the user know what's currently going on. Although this is difficult for web applications, modern technologies like RIAs or AJAX allow notifying the user what's going on at the backend. Error messages should primary be understandable by the user and not by the developer. Of course it should be possible to track the error for developers too, e.g. by dis-

playing details, but this should only be done on explicit request and not by default. The application should inform the user what he can do when an error occurs.

7. **Prevent input errors and clearly mark exits:**
   Instant notification of semantically or syntactically wrong values helps the user to correct these inputs directly without loosing the context. When content validation is done at the end of a longer input process, the user has to think about different values again and jump from error to error. It's also important to provide exit strategies for the user, e.g. when the user wants to create a new data record and the input dialogue appears, it should be possible to cancel this process and don't force the user to create a useless entry and deleting it afterwards.
8. **Shortcuts, flexibility, learnability:**
   User interfaces should have a design to be used from novices and experts simultaneously. On one hand assisted data input for beginners should help the user to understand the context of his entered data, on the other hand fast data input with shortcuts should be made available for expert users to save time.
9. **Online help and documentation:**
   Most web applications don't come with a printed documentation or a detailed user manual. But it's important to offer context-based help functionality to provide detailed information for the user, how to use the application and give examples what to input in form fields.
10. **Aesthetics:**
    Although aesthetics is very subjective, at least coloring and styling of the application should be appropriate and should not disturb the user when using the software.

Web-based applications for business applications

Nowadays business applications are re-implemented as web-based applications very often within company Intranets. Different approaches are available for these tasks to display rich data sets and allow users to manipulate these data. Business applications usually are used for complex tasks, such as flight planning, travel cost accounting, etc. Such tasks do have a large data model thus these applications were desktop applications earlier. When creating Rich Internet Applications,

which can handle these types of datasets, there are in general two ways, how to design and implement those: AJAX or Flash/Flex/Silverlight based applications that run in a browser plug-in. According to a Forrester Research paper [RIE08], Ajax is not the best choice when developing business applications. Ajax should be used for Ajax-driven mash-ups or “Ajaxified” HTML, but Ajax business apps don’t satisfy power users. Although Ajax is an improvement to simple page requests, as they were used in standard HTML applications, Ajax applications may also generate a lot of server roundtrips due to instant validation of input fields, etc. According to the research paper, standard HTML applications are often faster for users than Ajax applications. This may be based on the increased bandwidth of network and Internet connections. Therefore a complete page roundtrip may be “felt” faster, than a page that stays on the browser but has to validate after each input. Often the JavaScript interpretation process takes long and uses much CPU power on the client computer.

The paper [RIE08] describes that the Ajax applications really disappoint power-users of the business applications. One of the problems was that the complexity of input validation was very time-consuming. Developers had to reduce the real-time input validation compared to desktop applications, which were able to validate inputs instantly. Desktop application’s validation is very fast, because the executed code is compiled and faster done on the processor. JavaScript has to be interpreted, which consumes lots of time. Even if the validation was done of the server, the process produced a delay, because the data has to be sent to the server, processed over there and sent back to the client. On the client a notification has to be raised, if there was an error. These steps last very long and delay the power-user’s fast workflow. Another drawback of Ajax-based business applications was that they are not desktop independent. Although different Ajax frameworks use standardized methods, it cannot be guaranteed that the user will use the right browser in the right version. Therefore different workarounds have to be made and absolute desktop independence is not available.

The two major bottlenecks of Ajax applications are the commonly slow JavaScript interpreters and the slow access to the Document Object Model (DOM).

Finally Forrester recommends using large vendors’ ecosystems for Rich Internet Applications. This could be on one hand Microsoft’s Silverlight technology which fits in Microsoft dominated environments, or, on the other hand, Adobe’s

Flex and AIR technologies, which have a large lifecycle background, with access to Adobe’s LifeCycle data services and PDF capabilities.

Implementation of “Project Calculation”

Within Siemens it’s important to calculate the costs for projects quite accurate so that the price is reasonable for both, customer and producer. Therefore several factors are part of such calculations, which have to be considered during estimation. Not only manpower cost has to be considered, even risk management, material cost, travel cost, etc. are part of complex calculations.

For project leaders it has to be very easy to calculate the possible costs for projects to create an appealing offer to the customer. Basic data should be calculated and obtained by such a tool automatically, like hourly rates, travel costs for different countries, etc. The project executive should only fill in amounts of workload, travels, etc. to get a sum of costs very quickly. If the project leader would have to look for all these basic data by himself, it would take much longer to calculate and there might be much more errors when looking up such things manually.

The application “Project Calculation” was originally based on a Microsoft Excel Sheet with Macros and Visual Basic elements. The user could enter a fixed number of entries but the overall input process was very difficult and very limited. For example there were a predefined number of work packages the user could enter and the maximum project horizon was five years. Calculations that would exceed these limitations could not be calculated with that Excel sheet. There were also many other limitations with this technology: With Excel, when you’ve entered data, these data was kept within the sheet and could not be exported automatically. That means the data was always at the user who had the Excel calculation sheet. If the user didn’t extract the calculated values, nobody could use these data. Even if the user sent the Excel sheet via e-mail to colleagues, there could evolve incompatibilities because when they were editing the calculation there was no way to merge the edits together to a single, consistent calculation sheet.

Besides the data input in the Excel sheet was very complicated and without hints what to do for the user. Huge worksheets with several hundreds of rows and columns needed to be filled out and already entered data was hard to find in such giant datasets.

Now it was time to improve this application by using a completely different approach for such a tool. The requirement was to create a Rich Internet Application based on a platform with rich data processing capabilities and user interface elements with user-centric input processes. There were only very few constraints how to create, design and implement such an applications, that's why things were a little bit unclear during implementation and at some parts the desired development method "Agile Methods" didn't work out as intended.

Nevertheless the tool was developed with Adobe Flex, which delivers an Adobe Flash application as output and Adobe ColdFusion on the backend for service interaction. The application was deployed in two ways, on one hand as a browser embedded web application (as Flash file) and on the other hand as a desktop application, deployed as Adobe AIR application. AIR applications are capable of online/offline detection and the user can store data locally on the hard disk and synchronize with the server later, when back online.

Project description

The main purpose of the application was to provide a web application for project calculation that can be embedded into the Intranet of Siemens. So it could be accessed by anyone who needs to calculate specific costs for project cost estimation. Several different cost types need to be entered, e.g. personnel cost, travel cost, risk management, etc. All these values have to be calculated based on time (fiscal years) and work packages. The collected data then is used for documentation and controlling of projects and for transmission to continuative processes, e.g. bid presentation packs, risk reviews, etc.

Controlled by structured input assistance, the entry should be done as easy and intuitive as possible. Overview tables, tables as well as graphs, should help the user to keep an eye on the overall situation of the project during data input at any time.

It was important to develop the project calculation solution as a Rich Internet Application so that a high usability factor and a high transaction rate can be accomplished. It's also important that an XML export of all entered data is possible so that different other tools can work with the already calculated data too.

Another important requirement was that the application should be available in a multi-language user interface. That means the user should be able to switch the language of the user interface during data input at all time.

Purpose of the application

Goal of the application is, to allow easy setup for project calculations. Unlike the current application, based on Microsoft Excel, only the currently needed input fields should be displayed and allow data input. The Excel sheet always shows all possible data input fields, which is very confusing for the user. Therefore intelligent dialogues should only display context-relevant information. This is an important advantage in terms of usability for the user and should prevent errors in data input (because the data will be entered in the right fields).

The application should be web-based so that the entered data can be stored on servers and not only within the Excel sheet. This allows later analysis of overall project calculations. Due to central access within the Siemens Enterprise Portal access for all (authorized) employees is granted. It's also possible to define multiple users for one calculation. That means is possible that several persons may look at the latest and most up-to-date version of the calculation simultaneously. It's also possible to provide dashboard functionality. With the integrated dashboard statistic reports can be displayed with charts and diagrams for better visual understanding. In a later version, dashboard functionality for cross-project analysis will be a feature for the headquarter to compare the performance between projects.

The complexity of the data structure for the calculation requires process-driven user guidance. Pure database-table-driven mask applications don't have a high user acceptance rate and won't be a clearly arranged way for user input and data storage. By process-driven user guidance a mixture of expert-entry (as in Excel) and assistant-driven user input (as in a shopping cart application) is meant. Such guidance allows different entry paths for the user and not a single-path entry where several steps have to be completed before another can be started. Another important factor is that the user can have an overview over already input data at any time. Instantly available charts with key performance indicators help to summarize the project costs.

A selection of use cases of the application

Use cases are used in software development processes as a certain method to describe functionality of the application. Use case models are normally created using UML (Unified Modeling Language). Typically a use case is the summary of a UML diagram with a table, describing the entire use case. A typical schema for a detailed description of a use case contains the following properties: [HIT03]

- Use case name
- Short description / Summary
- Preconditions
- Post-conditions
- Exceptions
- Post-conditions when exceptions occur
- Actors
- Trigger
- Standard procedure
- Alternative paths

Most of the time, not all of these entities are available or required, but it makes sense to always use a template for use case descriptions with all entities for a better standardization of document templates.

The entire application consists of several functionalities, which are shown in the use case diagram in Figure 8:

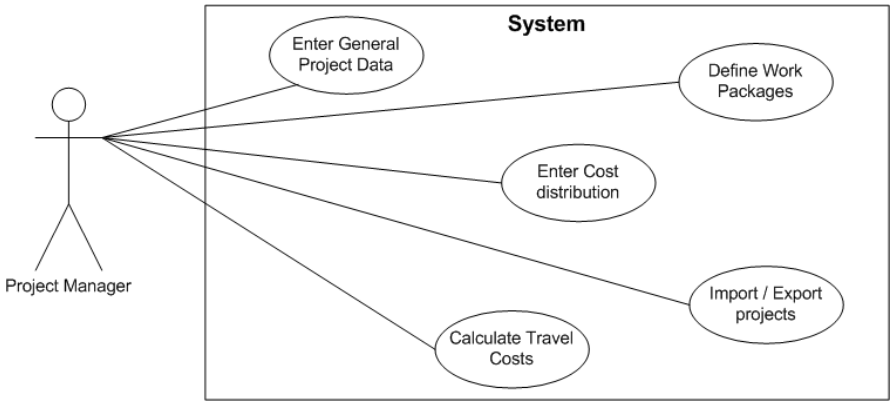

Figure 8: Basic use case for the ProCalc application; describes the main functionality of the ProCalc tool

Most of the functionality is described very basically in the use case overview above. For example the use case “Enter Cost Distribution” (Figure 9) could be split into several other use cases: Enter Personnel Costs, Enter Material Costs, and Enter Ancillary Costs. In the picture below, this generalization is displayed to detail the depths of the Use cases. For example Figure 10 shows the use case “Enter Personnel Costs”.

Table 4 and Table 5 below describe two use cases in detail.

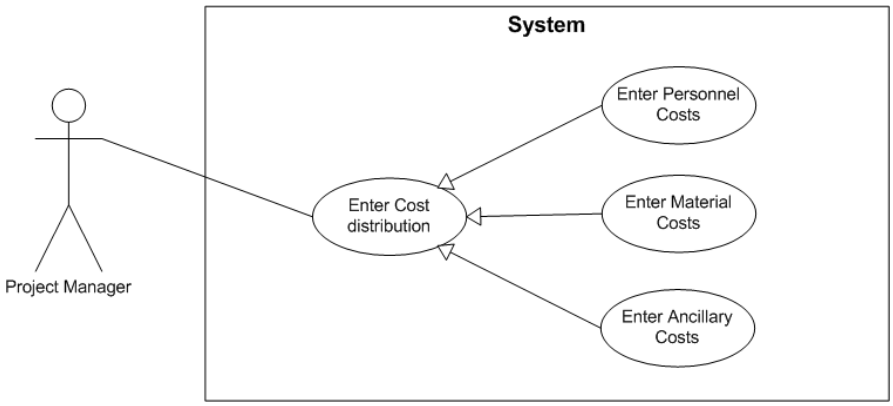

Figure 9: Generalization of use case "Enter Cost Distribution"

Use case: Enter Personnel Costs

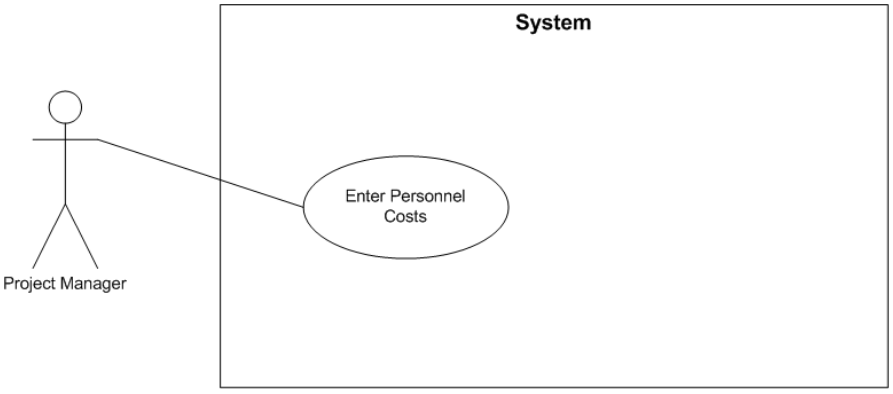

Figure 10: Use case "Enter Personnel Costs"

Use case name	Enter Personnel Costs
Short description	The project leader adds a dataset for the later calculation of project costs. This dataset represents personnel costs, which consists of entities like: fiscal year, working package, country and cost group or partner company, quantity, date, sum, description. The user can save the entry and delete or modify it afterwards. During data input, the sums of the entire project should be recalculated automatically for a better overview of the consequences caused by this dataset.
Preconditions	The system has loaded all necessary code tables (countries, fiscal years, etc.) The user has entered a lead country in the general project data settings. The user has entered working packages.
Post-conditions	New dataset for personnel cost is saved. The sum is automatically added to the entire project sum and also to listener variables, which are using values of the sub-sums of personnel costs.
Exceptions	-
PC when exceptions	-

Actors	User (Project leader)
Trigger	The user opens the window for adding personnel costs and clicks on “New Cost Entry”.
Standard procedure	1. The user clicks on “New Cost Entry”. 2. The user selects a working package. 3. The user selects a fiscal year. 4. The user selects a country. 5. The user selects a cost group. 6. The user enters the number of working hours or man-days. 7. The user enters a description for this dataset.
Alternative Paths	*Alternative 1:* 4. The user selects “external assignment” 5a. The user selects an existing partner company from the list of partner companies 5b. The user clicks on “Add Partner” to create a new partner company. 6. The user enters an hourly rate for the partner company. 7. The user enters the number of working hours or man-days. 8. The user enters a description for this dataset. *Alternative 2:* 6. The user chooses to add a constraint for his selected values for country, fiscal year and working package. He clicks on the number, which currently represents the multiplier between hours and manday and enters a new number.

Table 4: Use case description for use case "Enter Personnel Costs"

Use case: Edit a Personnel Cost Entry

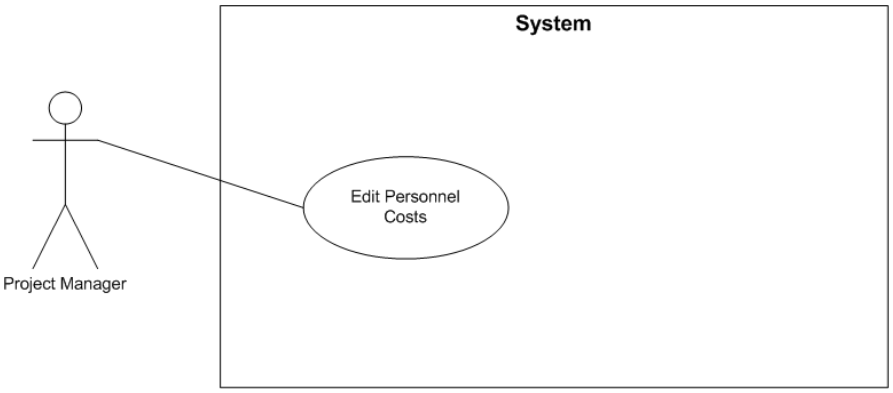

Figure 11: Use case "Edit Personnel Cost"

Use case name	Edit Personnel Cost
Short description	The project leader selects an already existing dataset and modifies parts of the properties. Afterwards the new generated sums are automatically distributed through the overall project data.
Preconditions	A dataset is available for editing.
Post-conditions	The dataset has been updated. The new generated sum is automatically updated within the entire project sum and also distributed to listener variables, which are using values of the sub-sums of personnel costs.
Exceptions	-
PC when exceptions	-
Actors	User (Project leader)
Trigger	The user clicks on an existing entry (of type “Personnel cost”) in the list of cost entries
Standard procedure	1. The system reads the selected entry from the list and gets all object metadata. 2. Based on the characteristics of the objects (whether it

	contains fiscal year, country and cost group, or fiscal year and partner company, the drop down boxes are preselected with the values of the objects. 3. Depending on the settings for fiscal year, country and working package respectively fiscal year, partner company and working package, a lookup is done for determination, if there's a constraint for the multiplier of hours to mandays. If so, it's displayed, otherwise the standard value is displayed. 4. All remaining fields are updated and prefilled with the dataset's values. 5. The user now can change / enter values as describe in the use case: "Enter Personnel Costs"
Alternative Paths	-

Table 5: Use case description for use case "Edit Personnel Costs"

Based on the two use cases above, the activity diagram in Figure 12 shows the possible usage of the Personnel cost dataset creation / manipulation. Multiple forks show the alternative paths for data input.

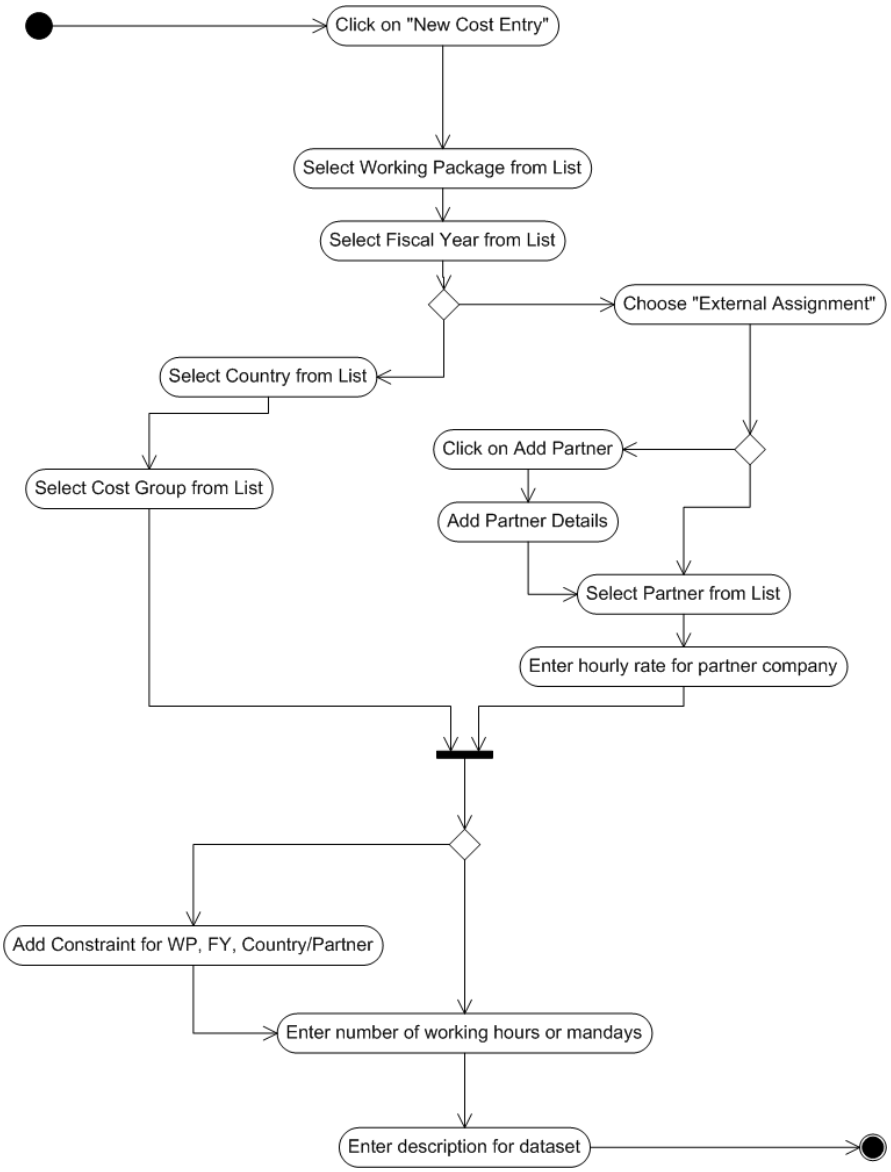

Figure 12: Activity Diagram "Add/Edit Personnel Cost"

Functionality of the application

When the application is launched, the user is asked for user credentials so the backend can check if the user is authorized to use the application. Next to this check, the backend searches automatically for already executed calculations or calculations by the user that are work in progress. A list with available and editable calculation then is displayed and the user can select the calculation he wants to edit. Next to this a list of version of the calculations is also available and the user can select, which version of the chosen calculation he wants to edit. This also includes the latest version of the calculation, called the “draft” version.

After that a connection to the backend is made to download the latest control information from the server. This control information contains current hourly rates for different countries, the list of countries, travel cost allowances … which are used in the calculation later. When the project is saved the first time, these values will be saved with the project and frozen. Otherwise the values would change over time, if the rates were changed, for example.

After the user selected a calculation or he decided to create a new calculation, the user can then choose what to do. The first step might be to launch the “general project data” (Figure 13) where he can input global data for the project. This might be the lead country (the country where the project’s headquarter resides), the currency, the first fiscal year, planning horizon, etc. These settings are needed in the further data input pages to display correct values, depending on the entered data. Several of these basic data can be chosen from a dropdown list for easy data input (e.g. there’s a list of countries).

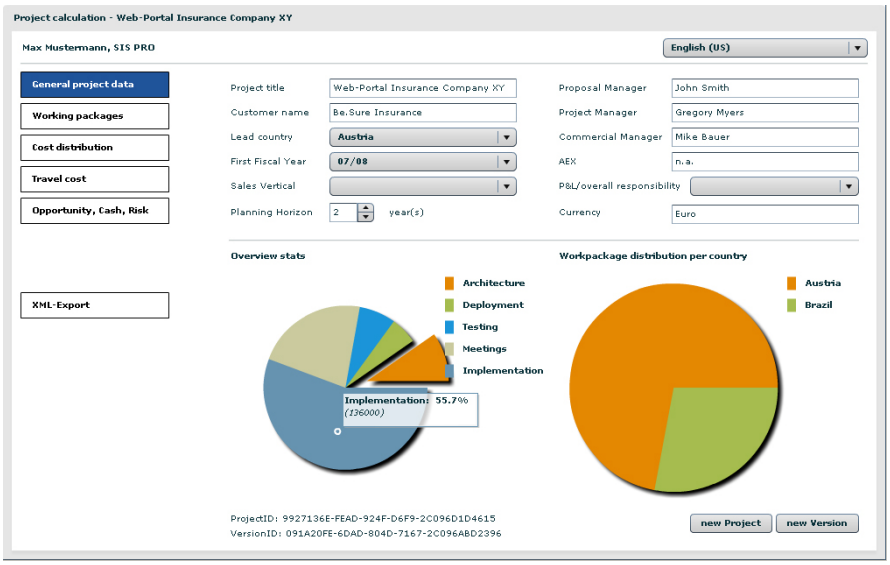

Figure 13: ProCalc Screen: General Project Data (some values are already entered, overview stats are displayed)

After the setup of the calculation project, there are several menu items available for the user to choose:

- Working packages
- Cost distribution
- Travel Cost
- Opportunity-, Risk- and Cash-Effect Management
- Financial Planning
- KPI Sheet
- Dashboard

These menu items provide instant access to all data that were already entered or allow entering relevant data for the calculation.

The menu “Working packages” (Figure 14 and Figure 15) allows the creation, modification or deletion of working packages. Working packages are used to classify different parts of the project, for example “Architecture”, “Design”, “Implementation”, … could be working packages. It’s also possible to assign individual colors to these working packages. The colors are shown then throughout the entire calculation for easy identification of different working packages.

Figure 14: ProCalc Screen: Menu Working Packages

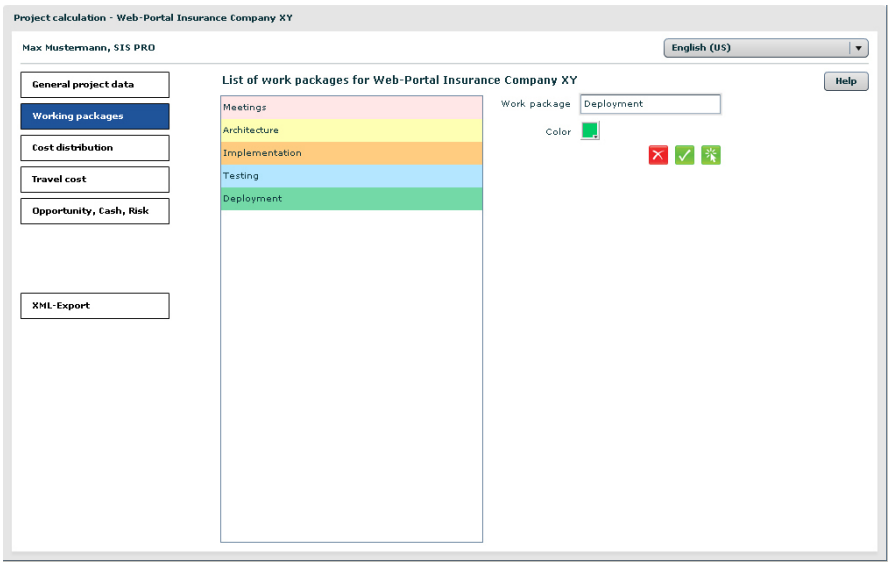

Figure 15: ProCalc Screen: Working Packages entered, Color coding

The menu “Cost distribution” (Figure 16, Figure 17, and Figure 18) holds all costs from the project except travel cost. It’s possible to enter manpower entries for employees in different countries. The user can choose working package, fiscal year, country and cost group and then enter the number of hours or workdays. The

rate for the manpower hours is defined from the combination of fiscal year, country and cost group and therefore calculated automatically. The user then can insert a short descriptive text for this dataset and save the entry.

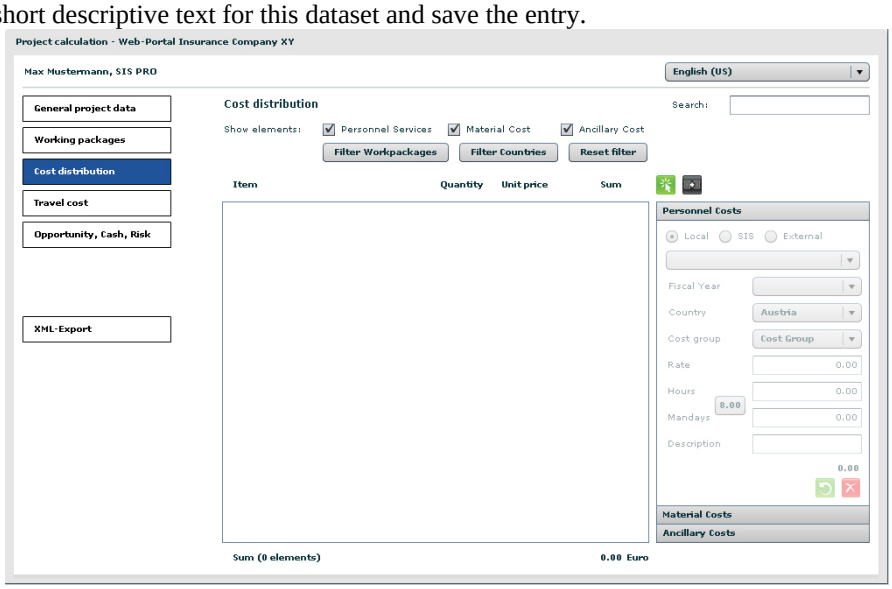

Figure 16: ProCalc Screen: Cost distribution editor (empty)

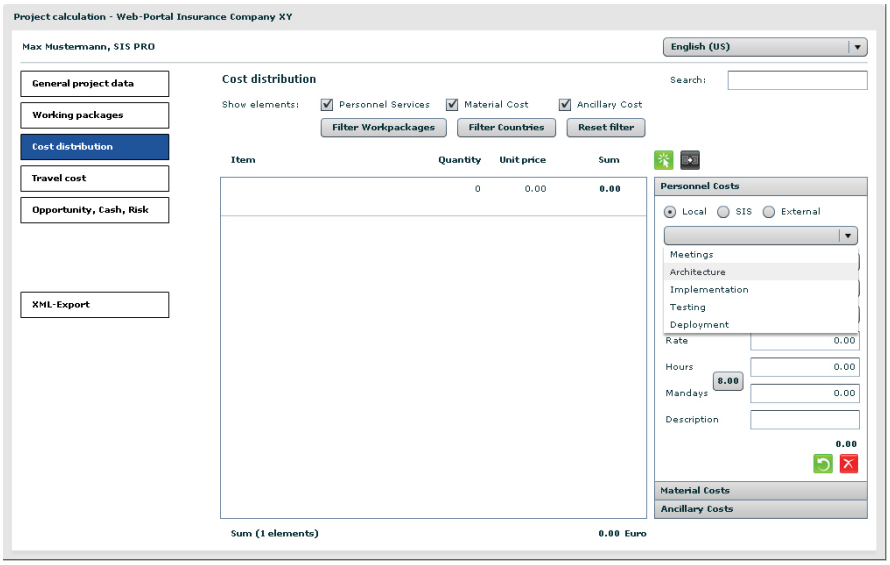

Figure 17: ProCalc Screen: Add Cost Entry "Personnel Cost", Selection of Working Package

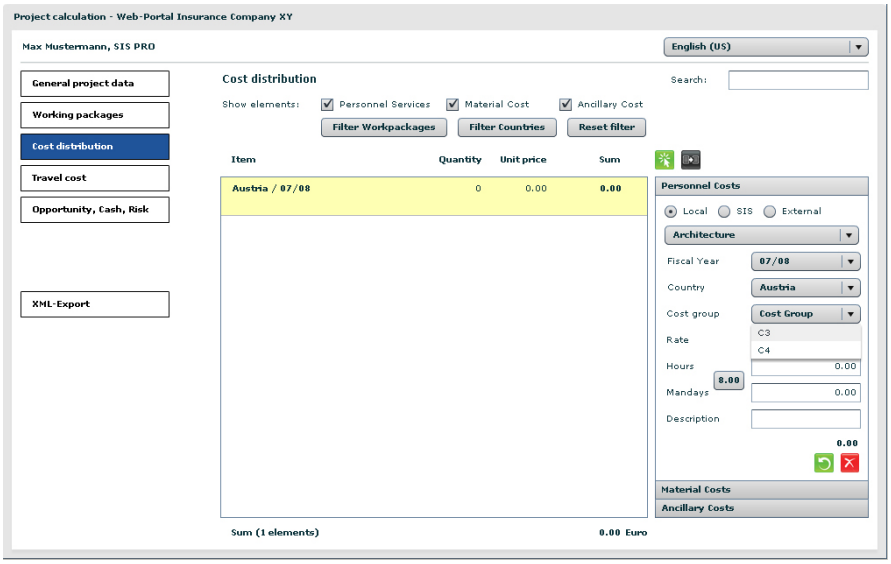

Figure 18: ProCalc Screen: Based on the selection of fiscal year and country, the drop down box for cost group gets populated with suitable values

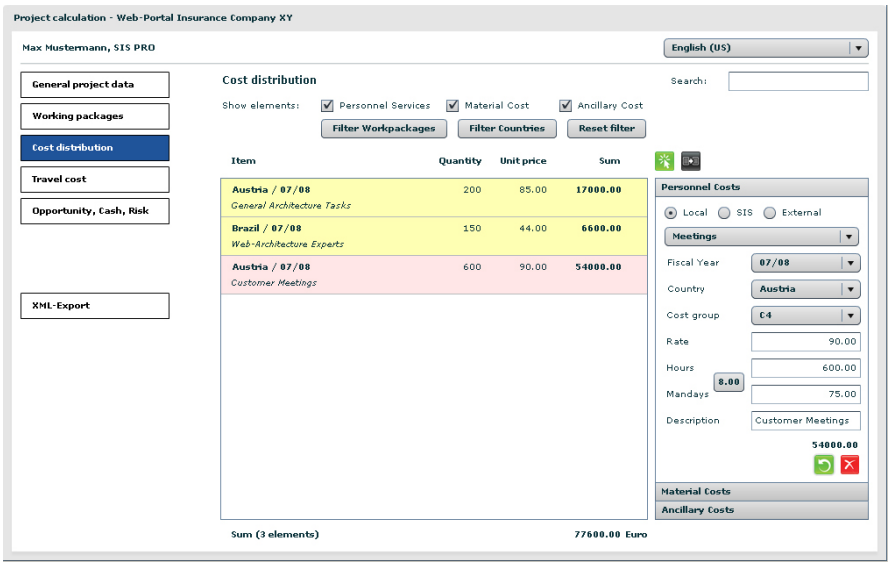

Figure 19: ProCalc Screen: Different Working Packages result in different color-coding in the list of all Cost Entries

Next to the data entry fields is a list with all already saved entries is shown (Figure 19) with a summary of the entered data. If the user clicks on the entry, the

data input fields automatically get prefilled with the selected entry and all the data for instant modification or deletion. It's also possible to duplicate entries when the user clicks on the item in the list and after that on the "save as new" button. It's quite easy to add different positions with only small differences doing that way. Not only manpower entries can be made, it's also possible to enter material costs and ancillary costs that may evolve in the project. The user can switch between these types very easily by clicking on the navigation accordion on the data input part of the application. Besides for external manpower (e.g. 3rd party purchases) it's possible to define partner companies (Figure 20) with custom hourly rates (Figure 21).

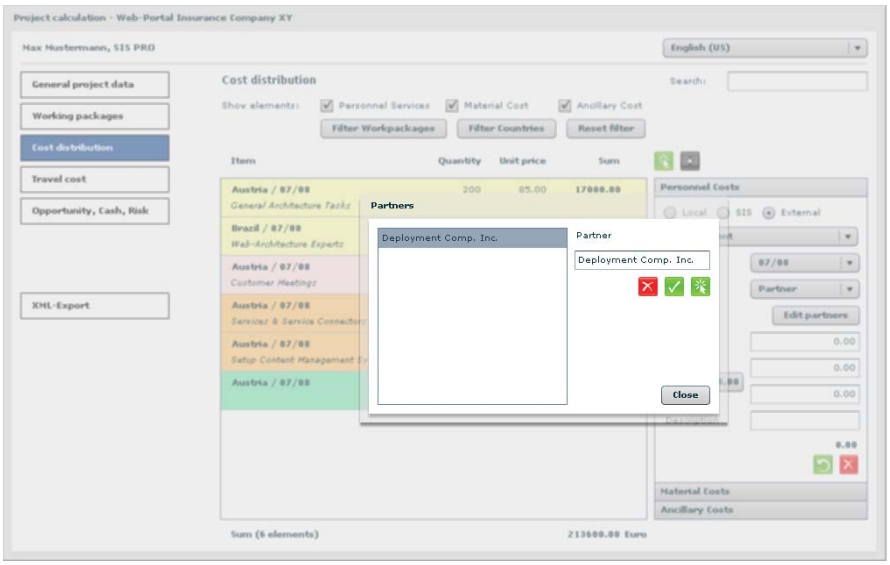

Figure 20: ProCalc Screen: Creation of a new Partner Company

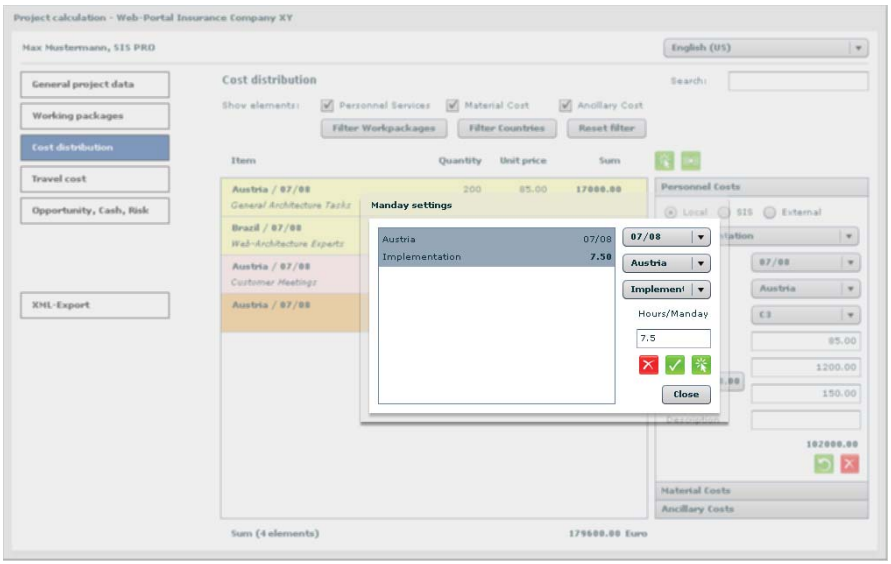

Figure 21: ProCalc Screen: Creation of Hours-per-Manday constraints for a given set of fiscal year, country and working package

Above the list with all entered positions an opportunity is provided to filter the data. It's possible to filter certain work packages (Figure 22) and countries and the result of the filter is immediately visible and the sums are re-calculated automatically in this view. Also a full-text search is available to search for certain entries in the list. The list is updated instantly while typing in the search field. Of course the filters and the search can be combined. Another possibility in this view is that the items can be sorted by clicking on the headers of the list. Sorting also works when filtering or during the search.

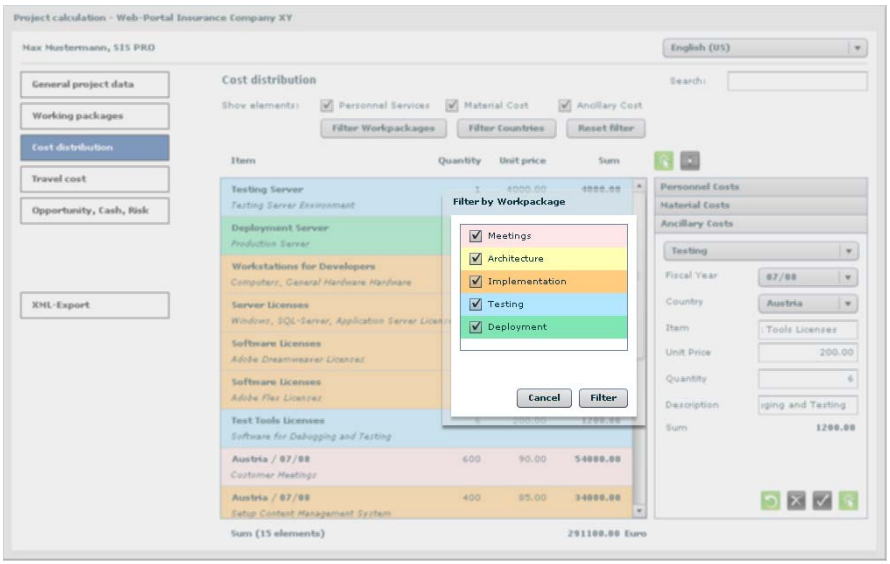

Figure 22: ProCalc Screen: Filter by Working Package

Within the menu “Travel cost” a calculation for possible travel cost can be started. Two different alternatives are available for the calculation. The first (and easy) alternative is that travel costs are proportional to manpower cost (Figure 23). There are three different input fields, where a percentage can be entered: for manpower cost within the own country (defined as lead country), international resources within Siemens and external partners. Next to the percentage input fields, the sums of the three manpower costs are displayed for a better overview. The user also can enter the number of days where travels may occur, so an average sum for each travel day is calculated.

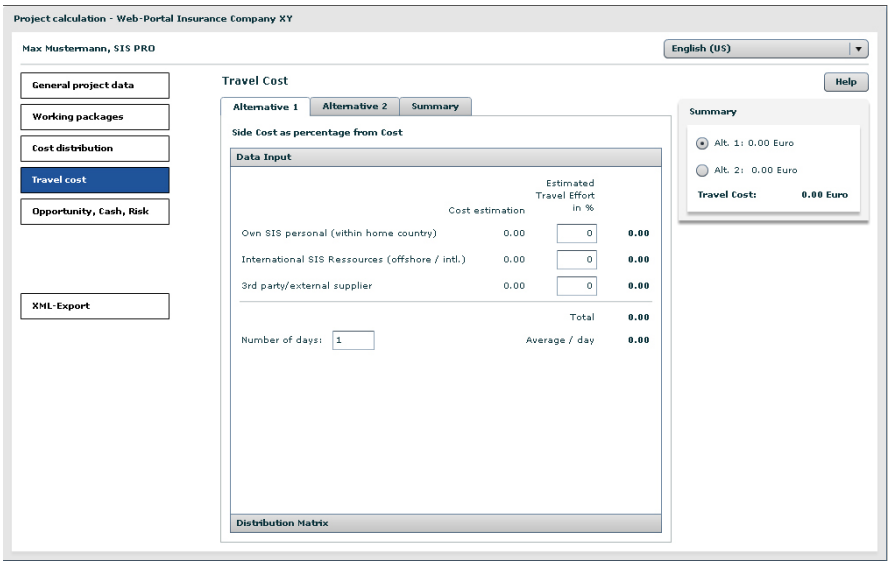

Figure 23: ProCalc Screen: Travel Cost entry, Alternative 1

The second alternative (Figure 24 and Figure 25) is far more detailed: here it's possible to enter exact values for several different parts of travel costs. First of all the travels per month and the number of traveling people can be entered. Depending on the part of travel cost calculation (home country, international assignments, external partners) different input fields appear. Several factors can be entered: flight costs between countries, rental cars, driven kilometers, railway, accommodation, public transportation, etc.). Based on these values again a sum for this alternative is calculated.

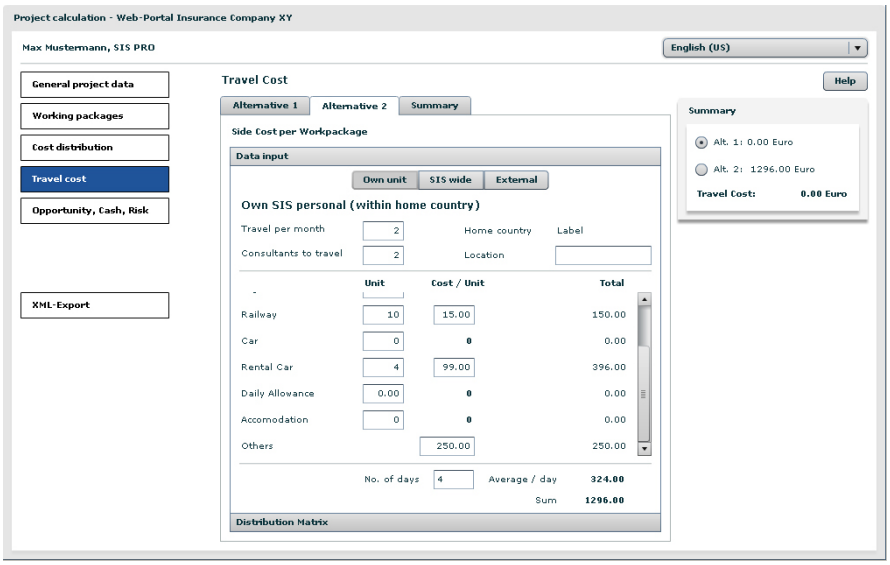

Figure 24: ProCalc Screen: Travel Cost entry, Alternative 2

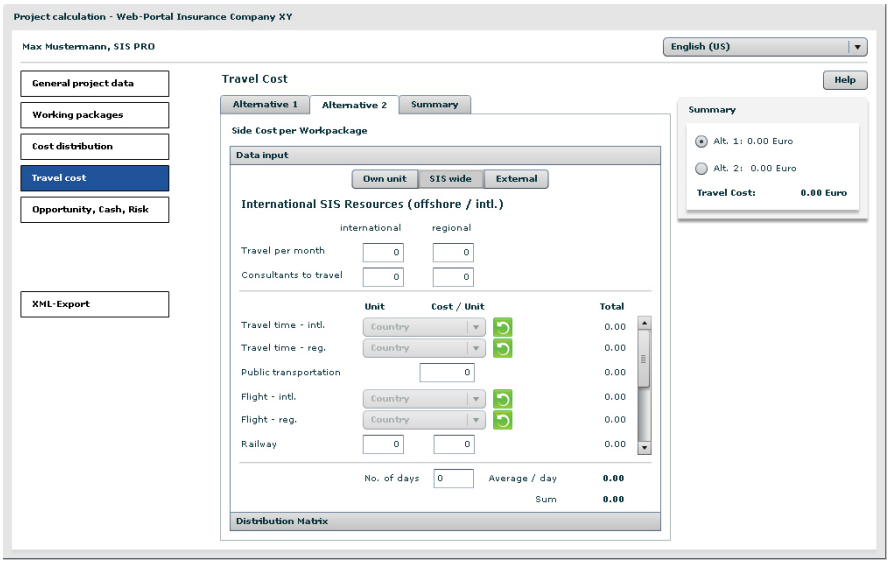

Figure 25: ProCalc Screen: Travel Cost entry, Alternative 2, international resources

For each of the alternatives a distribution matrix is also available (Figure 26). This matrix allows the distribution of travel costs over working packages and fiscal years. Below the columns and next to the rows the percentage sums are calcu-

lated automatically for a better overview how much money has been spent during certain years or working packages on travel costs.

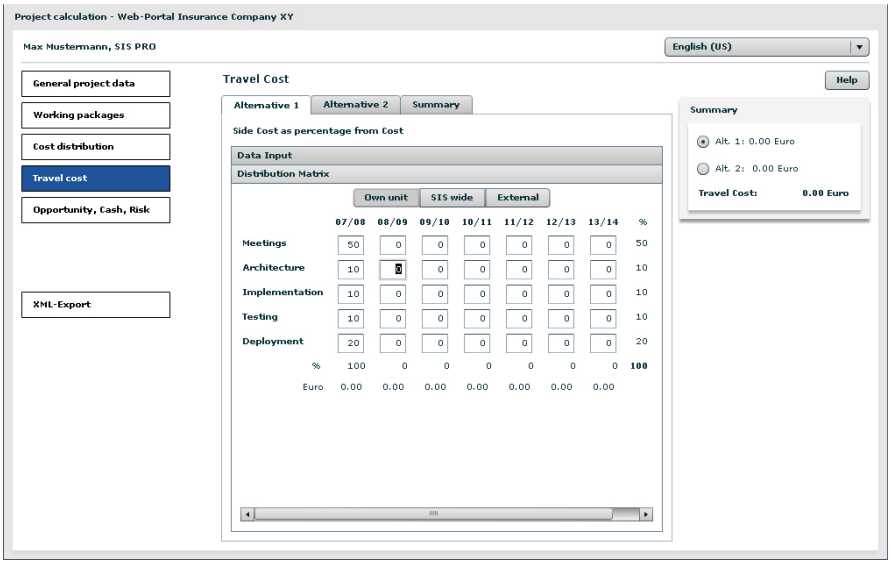

Figure 26: ProCalc Screen: Travel Cost distribution matrix

The next menu item is “Opportunity-, Risk- and Cash-Effect Management” (Figure 27). The user finds a list with entries of these types categorized within an accordion navigation. Below buttons are available for manipulation of these items. When editing or creating an item, a popup dialogue appears where the user can enter values for the current item (Figure 28). This popup window is also separated in multiple steps for a better overview and input guidance. For risk management a risk category has to be selected or a new one can be created for this single item. A risk owner person can be identified and a descriptive text can be entered. In the next step a “due date” can be selected for this risk item and the gross risk (in total) can be entered. The probability (in percent) has to be provided by the user and the weighted risk will get calculated. It’s also possible to select the risk impact (from minor to extreme) and the degree of implementation from drop down lists. In the last (optional) step it’s possible to split the risk to fiscal years by sum. That means it’s possible to move risks to certain years during project time for better calculation, when and how much money is used.

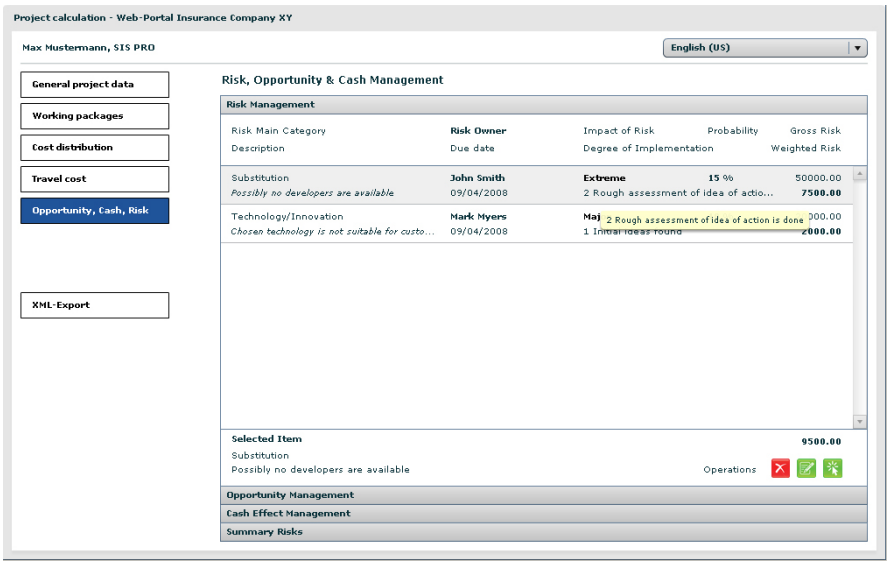

Figure 27: ProCalc Screen: List of entered risks

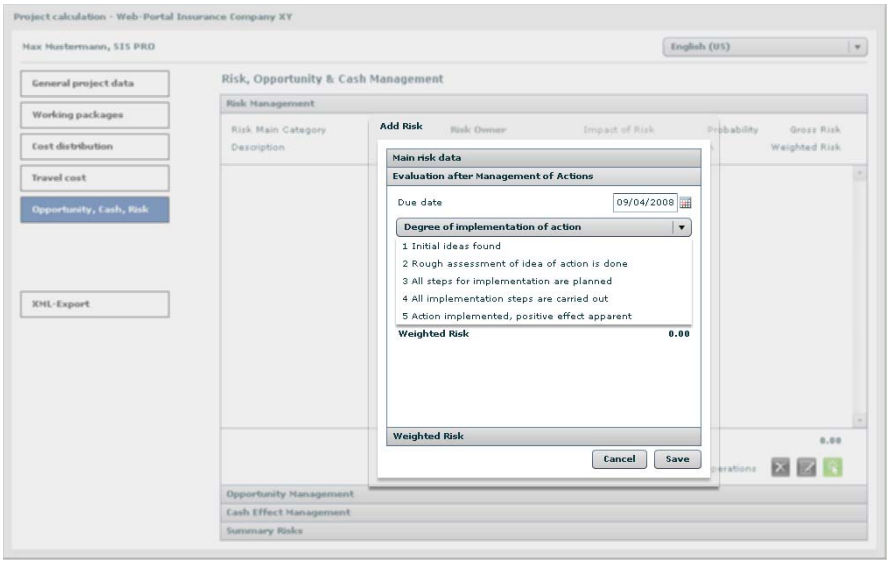

Figure 28: ProCalc Screen: Window for adding a Risk item

The same functionality is used within Opportunity- and Cash-Effect Management.

The menu “Financial Planning” provides the opportunity to enter parameters for depreciations of good, per fiscal year.

Finally the menu “Key performance indicators” the user finds a complete listing of all entered values in the project and all calculated values. Several performance indicators are displayed and aggregated by work package, cost types, and fiscal years. These performance indicators will be displayed within a second browser window as a popup in HTML format.

The last menu entry “Dashboard” offers detailed analysis for all entered and calculated data. The user can switch between different views. Work packages and fiscal years are variable parameters and can be shown in detailed views. The following reports are available:

- Costs per work package
- Costs per fiscal year
- Costs per work package and fiscal year
- Manpower per work package
- Manpower per fiscal year
- Manpower per package and year

These reports then are displayed in different charting views like: line charts, bar charts, or pie charts.

Used technologies

Due to the requirement that the application should be done as a Rich Internet Application, there were only a few technologies that can handle this requirement: AJAX, Adobe Flash or Microsoft Silverlight. Because Microsoft Silverlight is not yet deployed on many client PCs and AJAX does not allow automatic generation of vector-based charts for data visualization, the choice for the technology was Adobe Flash. In particular when talking about Flash, the Adobe Flash Platform is meant. Flash is only the output (as SWF file) of the authoring system for Rich Internet Applications named Flex. Flex (and its corresponding IDE FlexBuilder) delivers a huge set of reusable components for user interaction and service access. There are more than standard GUI elements like buttons or lists available; also more complex elements for navigation like accordions, view stacks or trees are available. Besides Flex is equipped with many interaction possibilities like web services, remote objects, HTTP calls or REST services, all these features are easy

to access for the developer. Flex is not only suitable for service-oriented architectures. It's also suitable for event driven architecture. The programming language of Flex is ActionScript 3, which is based on ECMA-Script 262 and therefore has all the capabilities for "eventing". Since the introduction of FlexBuilder 3 a new technology emerged developed by Adobe, called Adobe AIR. AIR stands for Adobe Integrated Runtime and is a cross-platform runtime environment for applications. Rich Internet Applications developed with Adobe Flash, Adobe Flex or Adobe Dreamweaver can be exported as AIR applications. These packages can then be installed to computers locally on to the hard disc. AIR applications don't need to be based on Flash: in fact AIR combines the strength of HTML and JavaScript (AJAX) with the power of Flash / Flex. The AIR runtime acts as a special version of a browser but has access to the local file system and does not run in a security sandbox as the Flash player does. AIR provides interfaces to the file system, direct database access, PDF generation, etc. Besides these special functionalities Flex applications can be ported to AIR directly without modifications. This advantage is quite big because existing Flex applications can then without reimplementation easily be ported to AIR applications and additional features like local file system access can be added quite fast to the application.
Another reason for the choice of Flash as deployment technology was the huge penetration of the Flash player on internet-enabled client PCs. Within Siemens all standard PCs have the latest Flash player version installed. But even outside Siemens the Flash player installation rate is at 98.8%, which means that nearly all internet-enabled PCs can run Flash content [ADO08].

On the backend, which has to deliver the data via web services, the decision was to use Adobe ColdFusion 8 as server technology. ColdFusion allows fast database interaction and quick web service deployments and it's used in the author's department frequently. There's also a specialized binary connection for Flash and ColdFusion, called RemoteObject in Flex, which speeds up the communication between client and server due to shorter message lengths. This connection uses AMF (ActionScript Message Format) and allows calling components or Java classes directly from Flex without exposing methods as web services.

The backend administration is based on an open-source content management system named "FarCry". FarCry is based on ColdFusion and acts as a web application framework. For developers it's very comfortable to set up new content types that can act as data storage. It allows automatic generation of administration

interfaces and saves a lot of time when developing web applications. FarCry can interact and set up different types of databases. In this application Microsoft SQL-Server is used.

Additionally to these technologies an interesting part is the communication between the backend and the frontend. As already mentioned above, “remoting” is used for the data channel between Flex and ColdFusion. A quite new part of these technologies are the Adobe LiveCycle Data Services ES. These data services allow a higher level of data integration in Rich Internet Applications. In the application LiveCycle Remoting is used, which uses the AMF3 protocol. This binary protocol allows the automatic mapping between JAVA objects (which are internally created in ColdFusion) and ActionScript objects. That means one can declare in ActionScript, to which of the remote classes this is mapped. When fetching or sending data between client and server, these will get serialized and deserialized automatically and casted to the right classes on both sides. There’s no need for the developers to cast the objects or build them together manually. To accomplish this mapping, ColdFusion components of the value object classes have to be generated. For example if there’s an ActionScript class Country.as, the corresponding ColdFusion component Country.cfc has to be created with the same public properties (or setter and getter methods). In ActionScript this ColdFusion component has to be referenced as “RemoteClass”. The data services automatically recognize if there are objects to transfer, which are typed with the remote class metadata attribute. If so, these objects are casted automatically to the right classes.

Project architecture: Before and now

Before: Architecture of the Excel solution

The architecture of the old tool was very simple. It was a plain Excel sheet, where different work sheets were used for data input. Entered data could not be extracted automatically; this could only be done by cut & paste. The work sheets were poorly designed and did not scale with the amount of entered data. All input fields were limited to a certain number of elements and could not be extended and not even reduced. The application was split into ten worksheets where data could

be entered and two additional worksheets with some lookup data for drop down boxes:

- **General Project Data:**
  On this sheet general data about the project can be entered. These data contains information about the project title, the project time frame and planning horizon, business sectors and the customer. There are also some fields for input, which are used for later display on bid presentation packs as proposal manager, etc. Additionally to these inputs there can be defined the work packages for the project. A fixed number of ten work packages can be entered to structure the project and the offer. Finally the project status can be set, whether it's before 1.0, 1.0 or after this version. The version 1.0 has to be frozen later for archiving.
- **Own unit & Summary:**
  Within this huge work sheet, which is a matrix between all work packages and all fiscal years with tables of cost groups, the user can enter the amount of work in hours, for each work package in each fiscal year of the project. It's not possible to add any details to the entries; this may be done in a separate work sheet or somewhere else. Additionally there are two free slots available where the user can enter material costs and three slots for ancillary costs. These costs are added to the sums which are calculated for each work package and finally for all costs for the own unit. Own unit means in general the own country where the project is located and accounted. Additionally to the sums of costs of the own unit, in this work sheet there's also the summary of travel costs for the own country and all costs of other countries and external assignments. These data will be entered in later work sheets.
- **SIS wide assignments:**
  The project leader can enter data for international assignments on this page. Additional to the input possibilities on the sheet "Own unit & Summary" the user can now choose a country for each cost group. Different countries and their cost groups have different rates, which have to be calculated correctly. A big problem with this sheet is that it's impossible to have manpower calculated with the same cost group in two different countries. And with the fact that no description for entries can be made, it's impossible to keep track of the entries, when they are manipulated so that this basic task could be accomplished (by changing the pricing codes for lookup tables).

- **External resources:**
  This sheet works similar to the sheet before. But instead of selecting countries, the user can choose from a list of partners, which may be involved in the project. The hourly rates for these partners can be defined manually.
- **Travel cost:**
  Travel costs can be entered with two different alternatives. The first and simple method is to define travel cost as percentage of manpower cost for each of own unit, international assignments and external resources. The second alternative is to input exact data for travel time, public transportation, rental cars, and accommodation and so on. For international assignments reference countries can be selected to calculate prices for flights and travel time in different countries.
  Additional to these main entries, there are matrices for each alternative that allow the distribution of percentages for work packages and fiscal year. So a better calculation can be conducted, which respects subtotals for fiscal years and work packages.
- **Opportunity, Cash & Risk Management:**
  Risk management is important for project calculation. In this work sheets not only risks can be entered, it allows also entries for opportunity and cash management. Risks can be classified by a risk category, which can be chosen from a drop down list. An owner and a description have to be entered too. Several other fields require input for a proper calculation like due date, degree of implementation of actions, probability, impact, gross risk, etc. The same fields are available for opportunity and cash effect management.
- **Project KPI Sheet:**
  This sheet displays a summary of all entered manpower, material and ancillary costs for each fiscal year and each work package. Below this listing a section called “Financial planning” is appended. The project manager can enter values for effects based on capitalization, e.g. depreciation, and effects on direct profit and loss.
  This sheet then displays the key performance indicators for these parts of the calculation.

In the Excel tool there were no statistics or reports generated to give an overview over the costs for the projects. It was impossible to find out how much money will be spent in one country or to get aggregated data for basic reports.

Now: Architecture of the Rich Internet Application

The newly developed Rich Internet Application “ProCalc” follows the basic architectural patterns of the Cairngorm Micro Architecture Framework [CGM] (which is described in-detail in a later chapter). Several “value objects” define the basic class diagram and how the data structure is built. The following picture shows an overview over the classes:

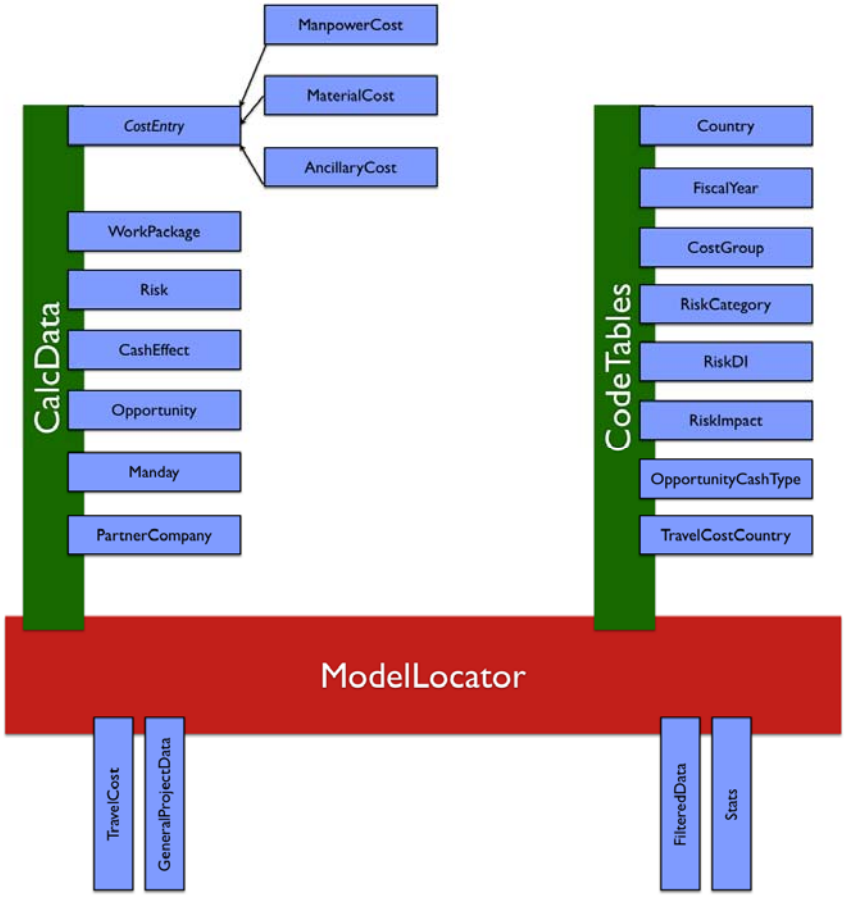

Figure 29: High-level overview of the ProCalc web application

Based on several blocks, the ModelLocator class is the central data storage for the entire web application. There are four major blocks within the model locator, which hold the data:

- **Code Tables:**
  The code tables are divided into several sub-classes, e.g. Country, FiscalYear, RiskCategory, etc. The datasets are transferred from the server to the client on project creation time. Due to the fact that these data may change over time, and therefore would affect the calculation without easy detection, why values have changed, these data are frozen, once retrieved and saved with the project. That means, once a project has been created, the code tables' data persist and cannot be changed.
- **CalcData:**
  Within the package CalcData, all relevant data for project costs are saved. On one hand the CostEntries (with its specialized occurrences ManpowerCost, MaterialCost and AncillaryCost) reflect the direct costs of the project, in terms of development time and material costs. On the other hand data such as Risks, Opportunities or Cash Effects can be entered and calculated.
- **TravelCost:**
  The next block is for travel costs. Travel costs are a complex and large structure of data, because there are several alternatives available in the calculation and each alternative should be saved. For each alternative there's also a matrix of percentages, how much of the travel costs should be accounted per work package, per country and per fiscal year.
- **GeneralProjectData:**
  Finally the class for GeneralProjectData contains basic project information, such as lead country, first fiscal year, project's title, responsibilities, etc.

There are two further classes within the model locator: FilteredData and Stats. The first one, FilterData is only used within the application for easy access to currently filtered data in all views. This class saves, if a user has defined a filter, e.g. to only show data from one single country.

The second class, Stats, is important because it has bindings on all values and creates statistics and reports as soon, as some values are changed. These data will also be transferred to the backend for later usage and cross-project reports.

How Agile Methods were used during development

At the beginning of project development the environment was completely different than at the end. At first, the customer wanted to have a working solution very quickly and also wanted the developers to find out, what he wants. That means there were no detailed requirements available. Therefore programming started as soon as a document was started, defining eventual requirements, to show some screenshots, how the project may look like. The author alone did the main part of the project. A colleague has developed just the backend connector. Therefore Agile Methods were not the way to go at the beginning of the project. During the development process, it became clearly that a more structured method of the development has to be defined. Agile Methods were chosen and used. One colleague acted as the Scrum master and Scrum meetings followed. Some principles of Agile Methods were changed to fit the project's size. Sprints were shortened to be weekly and the Scrum meetings were only made every second day. As already mentioned above, Agile Methods don't work for any project size with only a limited number of developers. However, it helped a lot to structure the workflow and find a better way of development than pure "hacking".

Finally Agile Methods were used for about 6-8 weeks during project progress. Several sprints were completed and meetings were held.

Cairngorm Micro Architecture

Cairngorm is an open source framework [CGM] for applications developed with Adobe Flex. Cairngorm provides a predefined set of classes and functionalities, to create well-structured applications in an extended MVC (Model – View – Controller) pattern. Software patterns are in general very useful in the most projects because it's easier to understand, how the application works, when viewing the source code and it's also valuable to have a predefined structure, how to write code. Therefore one can leverage on best-practice patterns and there's no need to reinvent the wheel.

Instead of the three-tier framework of MVC, Cairngorm provides an up-to six-tier architecture. These layers are:

- Business
- Commands
- Controls
- Model
- View
- Value Objects (VO)

These layers provide a great maintainability for the application. To describe these layers, the following picture (Figure 30) will be helpful:

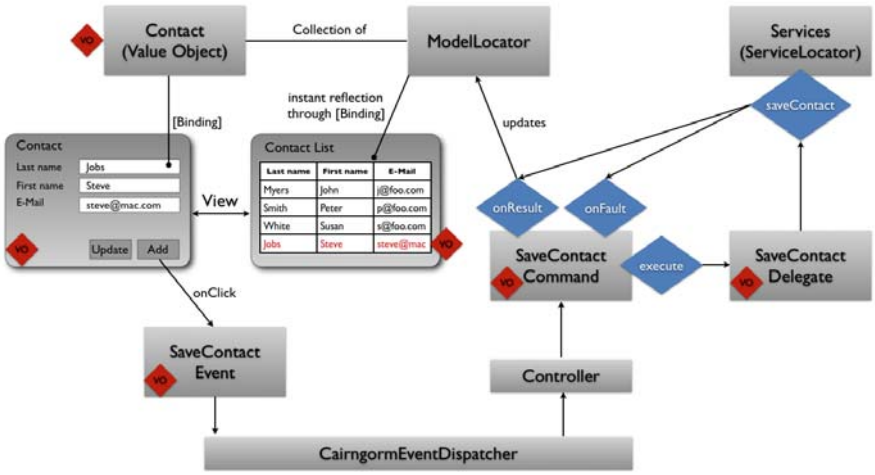

Figure 30: Cairngorm Micro Architecture Overview

First of all, an application has a view. A view consists of the user interface of an application, such as text input fields, buttons, list controls, panels, windows, etc. An application normally has multiple views, which are visible when they are needed. Within such views, data is displayed. This data comes from the model, in general from value objects. Value objects represent instances of data entries (similar to rows in a database table). The model contains all different value objects; in general, the model contains all data from the application in a structured format. ActionScript has the possibility to define variables, or even classes, as “bindable”. That means if these properties are changed, “bound” elements are updated too. Therefore one does not have to update the view, if the data changes, programmatically.

If the user now clicks a button or changes data (dependent on implementation), a custom event is triggered. The event then is dispatched to the CairngormEvent-Dispatcher. Cairngorm's architecture also makes use of the FrontController pattern. The FrontController acts as a central place where events are mapped to commands. If a certain event is fired, the mapped command is executed afterwards.

Custom Commands are derived from a Cairngorm interface and implement methods defined by the interface. One of the methods is "execute". This method is called automatically from the FrontController, when the suiting event has been dispatched. This method gets the dispatched event as a parameter and this event contains values specific for this command. For insert or update methods, the value object would be contained within this event. The command's execute method now creates an instance of the delegate class. This class is within the business package and creates the service call to the backend. The command class however also implements the Responder interface, which defines methods for result or faults of the service call. When calling the dispatcher, the command class passes itself to the delegate class as the suiting responder instance. Therefore the command class is responsible for creating the service call and also for handling responses and faults of the service. If the service call was successful, the result method will be called which parses the values returned by the service method. If valid results are returned, the model will be updated now by this method. And when the model is updated, all user interface elements, which are bound to the value objects in the model, are updated automatically. This represents the cycle of interaction within the Cairngorm framework.

As an example, let's have a look to a small application, made with Cairngorm (Figure 31). The example application consists of a small contact manager. It's possible to add and edit contacts, which can store first name, last name and an email address. The data will be stored to the backend via a web service call.

The main application contains references to the application's business object "Services", which acts as a service locator. It's quite easy to change the kind of service with this architecture, because the service locator acts as a central interface for backend communication. The main application also has a reference to the Controller, as the dispatcher for command execution. Within the main user interface, there are two view modules: ContactList and ContactEdit. The ContactList view (within the screenshot on the right side) displays a data grid with all Contacts,

which are already saved in the backend. The ContactEdit view (on the left side of the screenshot) allows users to create or edit certain contact items. When a user clicks on an entry in the list, the text input fields automatically get prefilled with the currently highlighted element values. If the user clicks on “Update” or “Add”, the ActionScript functions updateContact or addContact are called. Both methods create an instance of the value object type Contact and fill the data, based on the user’s entries. Then a SaveContactEvent is created which contains the contact data and a flag, if the entry should be added as new contact or an existing one should be updated. This event is dispatched by the CairngormEventDispatcher, which throws the Event so that the FrontController can handle them. The Controller class maps the event to a Command, which is the SaveContactCommand class. In the execute method of this command, the delegate is created and the command is added as the responder object. Within the delegate class, the responder methods for the service calls are defined (onResult and onFault from the command class). In the saveContact method of the delegate, the ServiceLocator is called to provide the instance of the service connection. With this instance, the service method is called. When there’s an answer from the service, the corresponding method is called (onFault or onResult). In the onResult method, the result from the service (the list of contacts) is written to the modelLocator. In the ContactList view, the data grid is bound to this ModelLocator property and updated instantly, because the ModelLocator is a “bindable” object, which allows instant change of all referencing components, if the property is changed.

Finally the application looks like this:

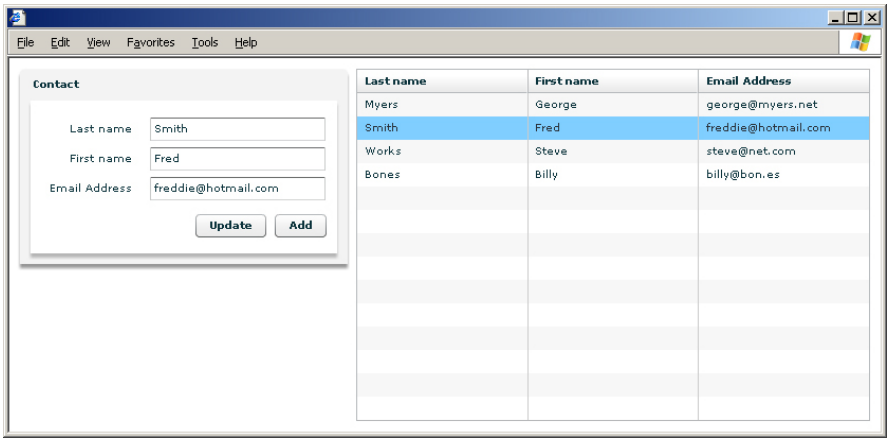

Figure 31: Example application: Contact Manager

Usability inspection for Project Calculation

After the implementation part of the ProCalc tool, a usability inspection has been performed. Detailed user tests are mostly too expensive in terms of money and time for companies, so usability inspections are commonly made. According to Jakob Nielsen “Usability inspection is the generic name for a set of methods that are all based on having evaluators inspect a user interface”. [NIE95]

Commonly for usability inspections 3-5 people are asked about the usability of the application. This can be done by using the application itself, or by showing those people printed screenshots of the application. If the screenshot method is applied, it’s important to let the testers use their fingertips as a pointing device. For example they should really press on the button symbol, when they want to press the button, which is shown on the printout.

For the usability inspection of the ProCalc tool, three people were used for the tests. All of them are employees at Siemens but they have different backgrounds: User 1 works for one year at Siemens and is primary a C/C++ developer for power systems. User 2 also works for power systems, but he also has to do a lot of management tasks. User 3 is a web developer for Rich Internet Applications. These three users were selected because of their different working areas but they all three users share the interest in well designed and easy-to-use applications.

A set of 30 printouts with several states of the application was shown to them. Also the users were asked 16 questions. During the test, a small story was explained to describe what the users should do. A protocol has been made for all decisions the users made and all questions, which were asked.

Before the inspection started, the purpose of the application was declared and what the application’s goal was. This was done without showing the first screenshot to let the user think about what he may expect. The process of using fingertips as pointer was described and that staying above elements would cause the interviewer to speak the tooltip, which would have been shown on the screen, if the application were used directly.

Together with Siemens’ Support Center for usability sixteen questions elaborated for the usability inspection. The questions were asked by the interviewer di-

rectly and sometimes the interviewer asked the users, what they might do during a certain state of the application and the users should explain in detail the purpose of the current screen.

The goal of the usability inspection was to find out the main issues of the user interface of the ProCalc application for a later improvement of the tool.

Results of the usability inspection

For the detailed result of the usability inspection only the most important questions and reactions are described here. The following table shows at first a description of the screen and the question and then shows the reactions / answers of the currently user. After the table, a summary is given, what the purpose of the application was and which things were recognized correctly and where room for improvement could be.

Screen 1: Entry screen of the application, "General Project Data"

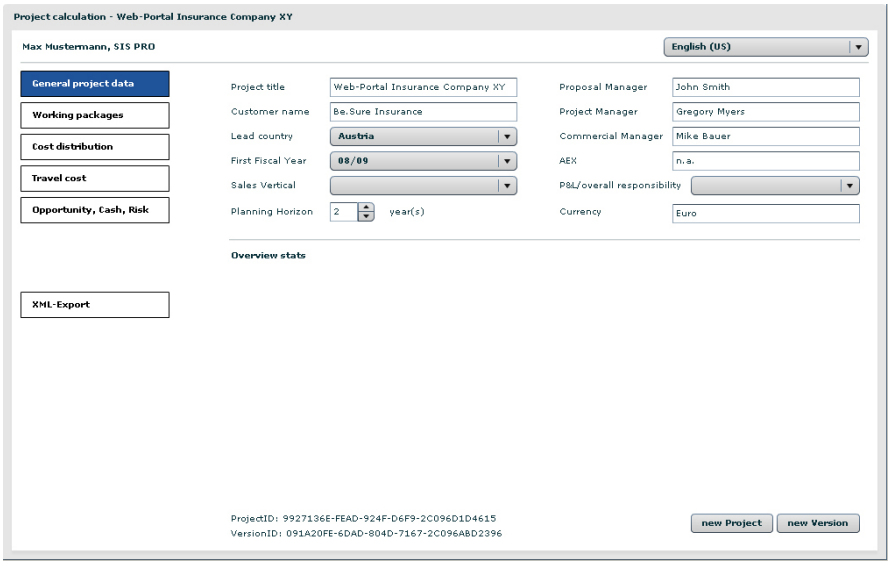

Figure 32: Usability inspection: Screen 1, "General Project Data"

Question: What do you see?	
User 1	User found:

	• All input fields for the general project data • Logged in user and his department • Menu on the left User didn't find: • Language select box Additional comments: none
User 2	User found: • All input fields for the general project data • Logged in user and his department • Menu on the left • Language select box User didn't find: - Additional comments: • Input field for currency should be a drop-down list instead of a free text input field
User 3	User found: • All input fields for the general project data • Menu on the left • Language select box User didn't find: • Logged in user and department Additional comments: none

Table 6: Usability inspection - Part 1

In the first screen, it was necessary for the user to find: the menu, the language select box, the information about the currently active user, and the data input fields for the general project data. One user didn't find the language selection box and one user didn't find the logged in user.

In general, the logged in user is a little bit odd to recognize, because the name was not specific to the interviewed person, so it may be difficult to identify the foreign name. For later usability inspections, the user name should match the test user's name.

Screen 2: "Working packages" with empty list of working packages, no input field was prefilled.

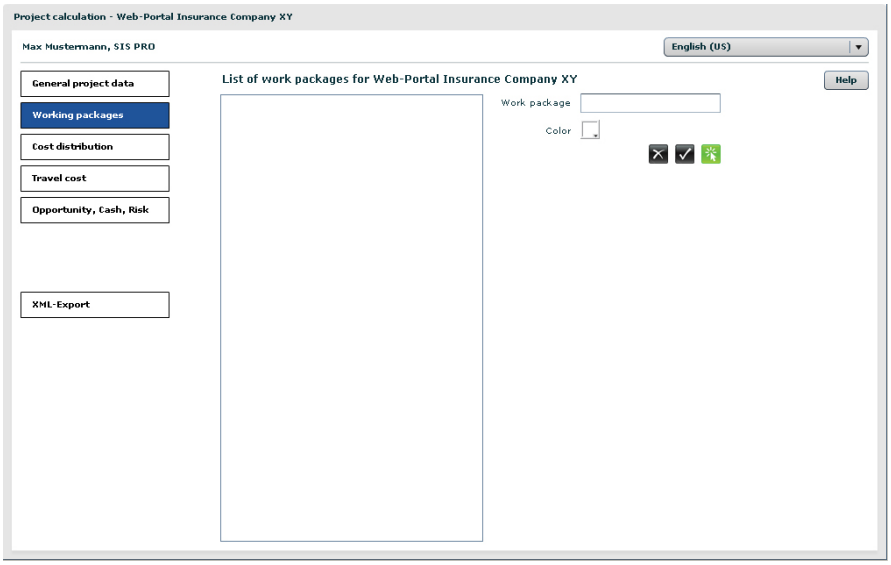

Figure 33: Usability inspection: Screen 2, "Working Packages"

Questions:

1. *What are the buttons for?*
2. *What is the color selection for?*
3. *How can a new working package named "Meeting" with color "pink" be created?*

User 1	1. User understood buttons correctly 2. After some thinking, user proposed that the colors might reflect certain departments or priority of the working packages. 3. User performed steps for creating a new working package correctly.
User 2	1. User did not understand buttons correctly. Didn't understand button "New", thought "Save" would act as new. 2. User thought, colors might be used for grouping working packages. 3. User performed steps for creating a new working package correctly.
User 3	1. User did not understand buttons correctly. Didn't understand button "New", thought "Save" would act as new. 2. Didn't understand color attribute, would click on "Help" to obtain

	information. 3. User performed steps for creating a new working package correctly.

Table 7: Usability inspection - Part 2

The buttons were mainly used for eye-candy only and because sometimes full-sized buttons didn't fit into the designated layout area because they were too large. Obviously users can't really interpret what these icons are for and therefore standard buttons should replace the icons again.

The color attribute in the working packages is used in the entire application. Every cost entry, which has an assigned working package, will always be displayed in this color. So users can easily find elements, which are grouped together. Because the users didn't understand the purpose of this color-coding, one has to think about removing this feature or describe the purpose in a text note next to the input fields.

There are two options for improvement of the application. Either the use of colors for working packages will be removed completely, because users don't understand them. Or their use will be explained in details, so that users know what they are for and they can use them correctly.

Screen 4: "Cost distribution" with empty list of cost entries. All input fields are disabled. *Screen 5 with active input fields, initial object in entry list with yellow background.*

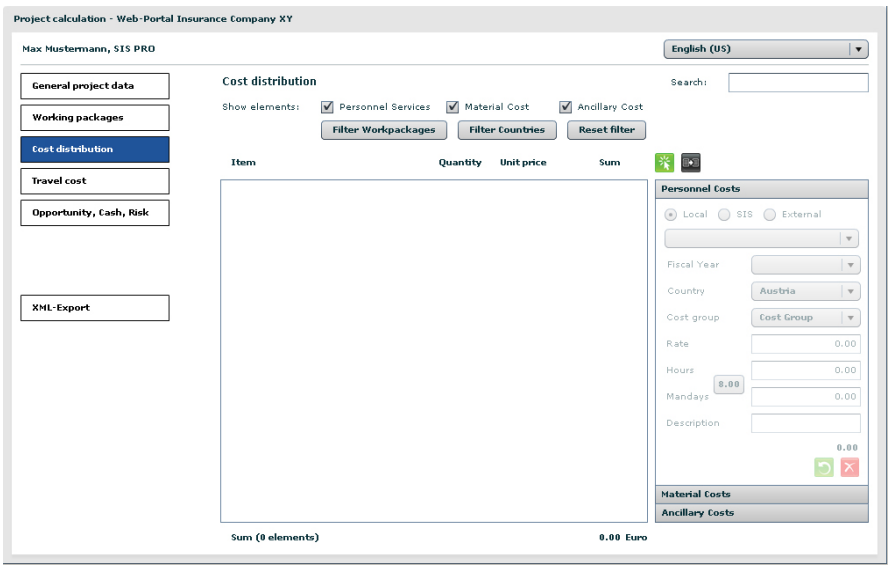

Figure 34: Usability inspection: Screen 4, "Cost distribution (empty)"

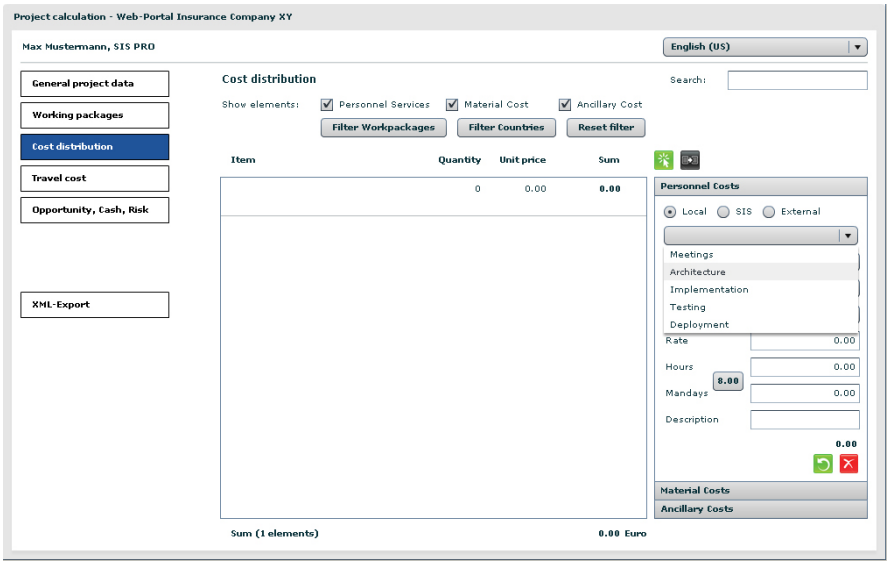

Figure 35: Usability inspection: Screen 5, "Cost distribution - select working package"

Questions:

1. *How would you create a new Personnel Cost Entry?*

2. Do you understand the buttons? *3. Why does the element in the list (Screen 5) have a yellow background?*	
User 1	1. The user clicked on the "New" icon, and then followed the correct sequence for adding a new personnel cost entry. 2. The user understood the icons, after the tooltip message was mentioned. 3. The user thinks, that's because it's in edit mode currently.
User 2	1. The user clicked on the "New" icon, and then followed the correct sequence for adding a new personnel cost entry. 2. The user did not understand the reset button and requested an explanation for it. 3. The user thinks, that's because it's in edit mode currently.
User 3	1. The user clicked on the "New" icon, and then followed the correct sequence for adding a new personnel cost entry. 2. The user thinks that the "Copy" button is for "Show element". 3. The user thinks, that's because it's in edit mode currently.

Table 8: Usability inspection - Part 3

As already mentioned above, the buttons are not as clear to understand, as they should be. And also the colors should be described in a better way already in the Working Package setup.

Screen 16: "Cost distribution" – Overview. All elements are entered; a list with 15 elements (mixed Personnel-, Material-, and Ancillary Costs) is shown.

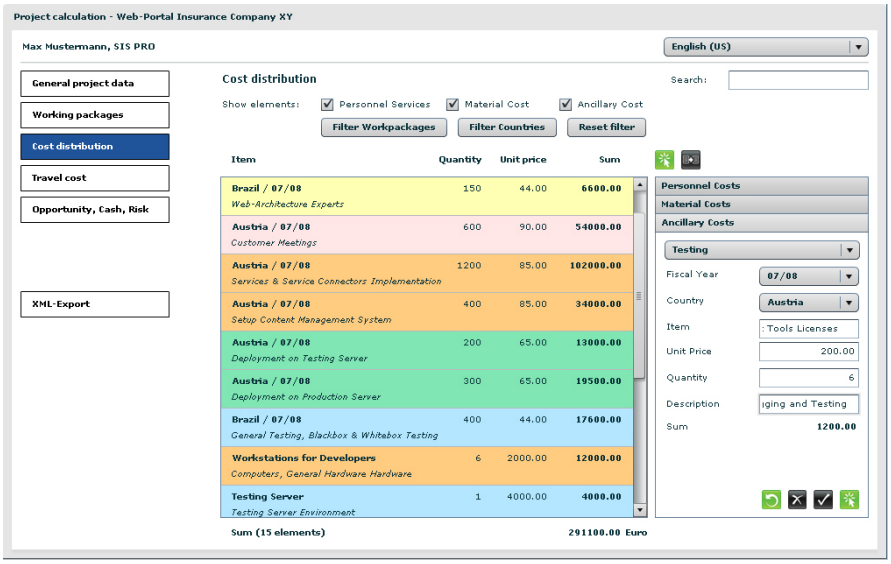

Figure 36: Usability inspection: Screen 16, "Cost distribution (full)"

Questions:

1. *Why are the elements shown in different colors?*
2. *How many elements have been entered? What's the overall sum of costs? (The user should recognize the footer with aggregated information)*
3. *How can you sort the elements, so that the most expensive element (most expensive unit price) is on top?*
4. *How would you act to only show entries from the working packages "Architecture" and "Implementation"?*
5. *How would you act to show only Material Costs?*

User 1	1. User did recognize that colors reflect the working packages. 2. The user found the footer and named the correct number of elements and the correct sum. 3. The user clicked on the header element "Unit price". The interviewer told him, that now the cheapest element is on top. The user clicked again on the header element "Unit price". 4. The user clicked on the "Filter Working Packages" button and selected all non-matching Working Packages in the new window and clicked on "Filter".

	5. The user deselected the checkboxes “Personnel Services” and “Ancillary Costs”.
User 2	1. User did recognize that colors reflect the working packages. 2. The user found the footer and named the correct number of elements and the correct sum. 3. The user clicked on the header element “Unit price”. The interviewer told him, that now the cheapest element is on top. The user clicked again on the header element “Unit price”. 4. The user clicked on the “Filter Working Packages” button and selected all non-matching Working Packages in the new window and clicked on “Filter”. 5. The user deselected the checkboxes “Personnel Services” and “Ancillary Costs”. Additional Comments: • The user missed filtering methods to filter by Fiscal Year. • The user lacks arrows next to the headers, when sorting was active. • The user missed toggle-buttons for select/deselect all in the filtering windows. • The user criticized that the data entry fields are not inactive when filtering is active; especially the data input field for “Ancillary Costs” was active, when the filter defined to show only Material Costs. • The user proposed to re-label the “Reset Filter” button as “Clear Filter”
User 3	1. User did recognize that colors reflect the working packages. 2. The user found the footer and named the correct number of elements and the correct sum. 3. The user clicked on the header element “Unit price”. The interviewer told him, that now the cheapest element is on top. The user clicked again on the header element “Unit price”. 4. The user clicked on the “Filter Working Packages” button and selected all non-matching Working Packages in the new window and clicked on “Filter”.

	5. The user deselected the checkboxes “Personnel Services” and “Ancillary Costs”.

Table 9: Usability inspection - Part 4

In general the users were satisfied with the current screen and they found instantly all of the questioned elements and values. Some minor tweaks can be made for the overall layout, but this is handled in the last question below.

Overall discussion.	
Questions: *1. Did you like the colors (for Working Package coloring)?* *2. Are you satisfied with the menu itself and its order?* *3. Do you miss anything, or is something there you didn’t expect?* *4. How would you switch the language of the user interface?* *5. What does the term “Search” mean in the Cost distribution window? Would the term “Filter” be more accurate?* *6. Do you have any other things you want to mention?*	
User 1	1. The user understood the meaning of the colors but he would choose different ones. 2. The user liked the menu and its order. 3. The user didn’t miss anything and didn’t find anything he didn’t expect. 4. The user would use the language selector on the top right corner of the screen. 5. The user thinks that with the search function he can search for “where / when / Working Packages / Description” within the Cost entry list. He likes the term “Search” more than “Filter”. 6. The users didn’t like the “Reset” button in the Cost entry dialogue. The button was not clear in its functionality.
User 2	1. The user understood the meaning of the colors but he would rather prefer not to use colors and have alternating row colors in white and gray. 2. The user liked the menu and its order. 3. The user didn’t miss anything and didn’t find anything he didn’t ex-

	pect. 4. The user would use the language selector on the top right corner of the screen. 5. The user didn't understand what he could search for. For filtering he would rather user the term "Free / Custom Filter" and the position of the filter box should be more prominent next to the other filters. 6. The user had several points he wanted to discuss: – He would like to have the cost entry list's header and footer to be more integrated into the list itself, e.g. by a border, so users can identify better that these things belong together. – The sum and number of elements label should be bigger to be found at first sight. – In the Working package view the user would rather place the input fields on the left and the list with working packages on the right. He thinks that it would be more natural to work "from left to right". He also would add a bigger description field for the working package, because "users would enter more detailed descriptions, if there's enough room. If the field is very limited, they only write very short descriptions". – The user expected to see overall sums for working packages already in the working packages view and the possibility to switch directly to editing mode from there. – In the cost distribution view the user also would change the order of the panels. He would event go that far to add a popup window for the creation and editing of entries so there's more space for the overview of existing entries. – He would like to have sorting possibilities for the fiscal year. – He would also increase the "description" field for cost entries, so users can input more detailed descriptions for the cost entries. – The user would like to have an additional field "comment" for cost entries, where project leaders can add additional information, why the entered e.g. this very number of working hours or they could enter, why they chose the selected rate according to

	current currency exchange ratio. – The user didn't like the icons at all, he'd rather prefer plain and easy-to-understand buttons.
User 3	1. The user understood the meaning of the colors and he mentioned that the colors are user-selectable so there's no problem to choose no color (white). 2. The user liked the menu and its order. 3. The user didn't miss anything and didn't find anything he didn't expect. 4. The user would use the language selector on the top right corner of the screen. 5. The user understood the meaning of the search function. He likes the term "Search" more than "Filter". 6. The user had several points he wanted to discuss: – The user requested a tighter integration of the footer to the cost entry list, e.g. by a border. – The user mentioned that the descriptive texts for input fields are not always understandable at first sight. – The user liked the tooltips for so many elements. – The user didn't really like the icons. In his mind one needs to get used to them, but with their tooltips they are ok. But the tooltips are very important to understand their meaning.

Table 10: Usability inspection - Part 5

Mainly User 2 described flaws and usability problems in the application. To act "from left-to-right" is for sure a good idea and should be tested with different users. The idea for using popup windows for data entry should be discussed very carefully. Opening a popup window may hide the content below and the user can't see the already entered values. But on the other hand a dedicated popup window would allow a more detailed and specific data input view and users may have additional controls for entering data.

The common question for a tighter integration of header and footer in the cost entry overview is easy to accomplish and in fact a good idea.

The users honored the idea of many tooltips because they helped to understand the purpose of some form elements. However the tooltips were indispensable for

the use of the icons and that’s not great usability. The icons in general should be revised or exchanged by common buttons.

The overall discussion with the users revealed that they were more than satisfied with the usability of the application. Of course they found some minor tweaks but compared to the “old” version of the project calculation tool (which was made in Microsoft Excel), it’s a huge step forward and helps the project leaders to calculate their project costs with a better overview and easier to understand.

Conclusion

During the development of the project calculation tool and during writing this diploma thesis, several remarkable things were detected.

To cope up with today's environments' and markets' needs, it's important to keep some things in mind: create stunning websites which are able to gain a high adoption rate using easy-to-use interfaces and be fast on the market. These two goals can be achieved, when developers and managers are able to use the right technology and software development process.

On one hand Agile Methods are commonly used in semi-large projects for a transparent development process and a reduced time-to-market. On the other hand and described in this thesis, Agile Methods are not the sole answer in product development and Agile Methods are not entirely scalable to small projects. Often rapid prototyping has to be done without strict rules and exactly defined development processes.

Nowadays most web-applications are plain HTML based. In the emerging need of business applications with rich datasets and high interactivity new technologies have to be adapted to satisfy users and customers. These so-called Rich Internet Applications rely on modern technology platforms and frameworks such as Microsoft Silverlight or Adobe's Flash Platform. Allowing users to work locally with data without permanent server roundtrips and server validation will result in more productivity and therefore in better results, because users have more time to think about the values they are about to enter.

Finally Rich Internet Applications have already started are going to be the upcoming standard for business applications, because they can visualize complex data into reports and react to data changes immediately. Adobe's Flex and AIR framework even allow taking the data offline, working with them on the road and synchronizing data, when one's back online. This is a great method to improve the workflow and to optimize the time, e.g. when someone is on a business travel.

Using these kinds of technologies is important for all developers to keep up with tomorrow's needs. – And finally: It's great fun :-)

Table of Figures

Bibliography

Books:

[NEW04] Eric Newcomer, Greg Lomow: Understanding SOA with Web Services, Addison Wesley, 2004

[SHO08] James Shore, Shane Warden: The Art of Agile Development, O'Reilly, 2008

[THO06] Dave Thomas, David Heinemeier Hansson: Agile Web Development with Rails, Second Edition, Pragmatic Bookshelf, 2006

[GAM95] Erich Gamma et al.: Design Patterns – Elements of Reusable Object-Oriented Software, Addison Wesley, 1995, 34th printing, 2007

[HRU04] Peter Hruschka et al.: Agility kompakt, Spektrum Akademischer Verlag, 2004

[MAR08] Tom DeMarco et al.: Adrenaline Junkies and Template Zombies – Understanding Patterns of Project Behavior, Dorset House, 2008

[LOT07] Joey Lott, Danny Patterson: Advanced ActionScript 3 with Design Patterns, Adobe Press, 2007

[ROY70] Winston Royce: Managing the Development of Large Software Systems: Concepts and Techniques, Proc. IEEE Westcon, 1970

[HIT03] Martin Hitz, Gerti Kappel: UMLWork, dpunkt.verlag, 2003

[NIE93] Jakob Nielsen: Usability Engineering, Academic Press Inc, 1993

Papers, Journals & Presentations

[RIE08] Stefan Ried, Ph.D. et al.: Ajax Disappoints Power Users Looking for Web 2.0-Style Business Apps, Forrester Research, Inc., 2008

[PUS08] Frank Puscher: Usability + Webdesign, Internet World Business Guid, 2008

[DIM05] Friedrich Dimmel: Service Oriented Architectures & Rich Internet Applications, Bachelor Thesis, University of Technology, Vienna, 2005

[ÖHM05] Peter Öhmans: The Rational Unified Process, Chalmers University of Technology, Sweden, 2005

[KRU01] Philippe Kruchten: What is the Rational Unified Process, IBM Developer Works, 2001

[APP08] Apple Inc.: Apple Human Interface Guidelines, 1992, 2001-2003, 2008

[KRO07] Kroneder Christoph et al., Agile Methods for SEM, 2007

Videos

- Ken Shwaber in Scrum et al.
  http://video.google.com/videoplay?docid=2531954797594836634
  retrieved on: April 16th, 2008
- Jeff Sutherland in Scrum Tuning: Lessons learned from Scrum implementation at Google
  http://video.google.com/videoplay?docid=8795214308797356840
  retrieved on: April 16th, 2008
- Several Cairngorm introduction videos by David Tucker
  http://www.davidtucker.net/2008/04/01/cairngorm-videos-available-as-flv-downloads
  retrieved on: May 20th, 2008

Web Pages

- [HOW01-1] Forrester Research - X Internet, Carl Howe et al., 2001
  http://www.forrester.com/ER/Research/Report/0,1338,11282,00.html
  retrieved on: April 16th, 2008
- [HOW01-2] The X Internet by Carl D. Howe with George F. Colony, Bill Doyle, Christopher Voce, Rebecca Shuman - Forrester Research
  http://www.forrester.com/ER/Marketing/1,1503,214,FF.html
  retrieved on: April 16th, 2008

- [OUT] Outsmart / Our Technology - Rich Internet Applications
  http://www.getoutsmart.com/rich-internet-applications.htm
  retrieved on: April 16th, 2008
- [IDC03] IDC: Rich Internet Applications, 2003
  http://www.adobe.com/platform/whitepapers/idc_impact_of_rias.pdf
  retrieved on: April 16th, 2008
- [SZA07] Scrum Alliance - Glossary of Scrum Terms, Victor Szalvay, 2007
  http://www.scrumalliance.org/articles/39-glossary-of-scrum-terms
  retrieved on May 9th, 2008
- [ADO08] Flash Player version penetration
  http://www.adobe.com/products/player_census/flashplayer/version_penetration.html
  retrieved on: August 7th, 2008
- [NIE] Heuristic Evaluation/ How-To Conduct a Heuristic Evaluation
  http://www.useit.com/papers/heuristic/heuristic_evaluation.html
  retrieved on: July 16th, 2008
- [NIE95] Summary of Usability Inspection Methods, Jakob Nielsen
  http://www.useit.com/papers/heuristic/inspection_summary.html
  retrieved on: September 17th, 2008
- [NIE00] Why you only need to test with 5 users, Jakob Nielsen
  http://www.useit.com/alertbox/20000319.html
  retrieved on: October 8th, 2008
- [NIE05] Heuristics for User Interface Design
  http://www.useit.com/papers/heuristic/heuristic_list.html
  retrieved on: July 16th, 2008
- [EVE00] Rational Unified Process
  http://everything2.com/e2node/Rational%2520Unified%2520Process
  retrieved on: August 6th, 2008
- [W3C] W3C Consortium / Web Services
  http://www.w3.org/2002/ws/
  retrieved on: August 28th, 2008
- [SOAP] W3C Consortium / SOAP
  http://www.w3.org/TR/soap/
  retrieved on: October 8th, 2008

- [WSDL] W3C Consortium / WSDL
  http://www.w3.org/TR/wsdl
  retrieved on: October 8th, 2008
- [ALL02] Jeremy Allaire – Macromedia Flash MX – Next generation rich client, 2002
  http://www.adobe.com/devnet/flash/whitepapers/richclient.pdf
  retrieved on: October 8th, 2008
- [SHN] Ben Shneiderman – Eight Golden Rules of Interface Design
  http://faculty.washington.edu/jtenenbg/courses/360/f04/sessions/schneidermanGoldenRules.html
  retrieved on: October 8th, 2008
- [SEM] Siemens Software Engineering Methods
  http://www.pse.siemens.at/apps/sis/ge/pseinternet.nsf/0/PK1A6844054A7E23F2C12569EE003A1A93/$FILE/QM_folder_deutsch.pdf
  retrieved on: October 8th, 2008
- [CGM] Cairngorm Micro Architecture
  http://opensource.adobe.com/wiki/display/cairngorm/Cairngorm
  retrieved on: October 8th, 2008