Twilm
ONELINER ABOUT THE PROJECT HERE

NTNU – Trondheim
Norwegian University of
Science and Technology

SPECIALIZATION PROJECT? =P
October 29, 2013
Team: Fredrik Persen Fostvedt and Martin Havig
Supervisor: Heri Ramampiaro

Acknowledgements

Thanks Mum!

Abstract

Skal brukes til å fange leserens oppmerksomhet Skal summere innholdet av raporten Bør være mellom 1/2 - 1 side lang Bør innholde: kontekst, mål, og hva som er blitt oppnådd. Også hva som er nytt / ny oppdagelse bør være med

Some awesome abstract, example:

This report will give the reader an insight into the details of the design, development and implementation of the task given in the course TDT4290 - Customer Driven Project, taught at NTNU - the Norwegian University of Science and Technology. The customer is Netlight and they have presented the group with the task of breathing new life into the console.

Web-applications these days are leaning against a mouse-controlled, web-fronted design. This has taken away much of the efficiency of power users, who have traditionally used terminal applications on a daily basis, and had the system in their fingers.

A hybrid web-fronted/console design would be a possible solution to this problem: The power user can make use of their full potential through a console whilst the objects are presented in the web-interface.

This is a proof-of-concept task, and all research done will be documented and used to argue for and against the solutions used and not used. Everything from the planning of the project startup and preliminary-study to the complete conclusion is described in this report.

The approach to investigate and solve this problem starts with a thorough study of relevant technologies, and how this can be made possible. The conclusion of this study allows us to create a system which showcases the real potential of our soution. Through this whole process we have a close work-relationship with our customer to ensure his desires and expectations for the project are met, and that our conclusions and findings boosts future research in this field.

Acknowledgments

Some awesome Acknowledgments

Vet ikke hva som er best, acknowledgments før eller etter abstract, tar det l8er

I hoved-kapitlene beskriv selve forskningen, og dens resultater. Inkluder nok teknisk detalj slik at leseren klrer å bedømme nøyaktigheten og originaliteten av arbeidet ditt. Denne beskrivelsen av forskningen bør være den største delen av avhandlingen din, hvordan den er delt inn i kapitler vil avhenge av arbeidet ditt.

Preface

Some awesome preface, example:

This report serves as the primary piece of documentation of our attendance to the TDT4290 Customer Driven Project course, hosted at the Department of Computer and Information Science at the The Norwegian University of Science and Technology, in the autumn of the year 2012.

Our customer was the Netlight consulting company represented by Peder Kongelf.

We would like to thank our supervisor, Zhu Meng, for his feedback on our work, and also Peder Kongelf for presenting us with the opportunity to work on such an interesting project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

Context; motivation for the project; problem statement; outline of dissertation.

I introduksjonskapittelet, forklar kort konteksten med og motivasjonen bak arbeidet ditt, forklar problemet som du prøver å løse og forklar hvorfor dette problemet er verdt å løse.

Wonsole is a student project under the TDT4290 course in IDI, NTNU. The project is intended to give all its students experience in a customer guided IT- project and the feel of managing a project in a group. Every phase of a typical IT- project will be covered. This report will serve as documentation of our work. This includes our work progress, the technologies we used, our research findings and so on. The introduction chapter is meant to describe the project, our goals and briefly the involved parties.

1.1 Purpose

1.2 Motivation

1.3 Context

1.4 Project

This is a proof of concept project. The underlying task is to research and develop a system where power users can benefit from a console. The concept aims to ease the workload of a power user who is working with object editing, and to see how the efficiency of a console might prove to improve the work. The power user is usually a user who often works with the system over a longer time, and is in depth familiar with the system. We will research already existing systems of this kind, and look at the possibilities and advantages of such a system in a chosen domain.

We have chosen a library as our domain, and this will be used to explore and test the concept. The library domain is chosen since it possesses potential for the existence of power users and multiple input forms which could be made more efficient through a console. This domain also opens the opportunity to test our system on for instance employees on campus, which is important for the proving of the concept.

Goals

1. Provide extensive documentation and a successful presentation of the end product.
2. Create a working prototype of a system where a scripting console is embedded into a modern web interface. The console should provide access to viewing and modifying the underlying data objects of the system's domain via a Domain Specific Language(DSL).
3. Investigate the ramifications of the added functionality, in terms of usability and technical aspects.

It is important to note that the report is in focus. It will be the cornerstone of the prototype, to not just ease further development, but also to amplify the reasons for the choices we make.

1.5 Project name

Project name is important project identificator. It should summarize main project goal or functionality. In real projects, this is often a trademark, or a name that reflects the name of the company. For our project, the main concern was to create a descriptor that reflected the root concept, namely incorporating a console into a web application.

We held a brainstorming meeting specifically to create a name for the project. Early in the process, we created a list of words that could describe our project functionality or goal. Some keywords:

- Master, Super User
- Console, Terminal, Command Line
- Web Application, GUI
- Internet, Networking
- Text, Keyboard

From these keywords we attempted to compile a list of candidate names:

- Console 2.0
- Wonsole
- Wensole
- Websole
- Werminal
- interCLI

After a discussion and a brief investigation into which names were already taken by other projects, we chose the name *Wonsole*. The project name alone can be a little confusing, so we added the subtitle: *The new web console for power users.*

1.6 Structure of Report

This report describes the development of bringing the scripting experience to the power user. The report is structured after the course of the project, and gives the reader insight into the development of the system.

Chapter	Description
Chapter 1	The introduction chapter introduces the problem to the reader, introduce the members and stakeholders of this project and explain the motivation for doing this project. After reading this chapter, the reader will be left with an overall view of the projects outline and goals.
Chapter 2	The Preliminary Study chapter describes the work and research done, and road taken to the choices we made. This includes technologies, methodologies, testing and version control.
Chapter 3	The Project management chapter deals with how the team will work to reach the desired goal. This includes the structure of the team, the plan for the project, constraints the team and project is put under and quality assurance.
Chapter 4	The Requirements chapter describes the backlog and rationale for this. Use cases for the system is also placed in this chapter together with user stories.
Chapter 5	The Test Plan chapter contains the approach for testing with the overall test plan for the project, and the test scheduling.
Chapter 6	The Architecture chapter explains the structure of the system, how it is put together and why it is put together the way it is.
Chapter 7 - 10	In the sprint chapters the reader can see how the product has developed during the time of the project. This includes planning, architecture, the implementation, testing and evaluation of the sprint.
Chapter 11	Evaluation chapter includes the team's view on the team dynamics, relationship with the customer, issues during the project, the planning and the quality assurance.
Chapter 12	Conclusion chapter sums up the project and describes the findings and reflections around this.
Appendix	The appendix contains tables, templates, the test cases and more.

Table 1.1: Structure of Report

Chapter 2

Preliminary Study

Contents

Review of relevant literature; review of similar software products or tools.

I "survey"-kapittelet presenter en oversikt over tidligere relevant arbeid inkludert artikler og evt. eksisterende produkter. Vurder kritisk styrker og svakheter av tidligere arbeid

This chapter is ment to outline the preliminary study of our project. This includes what our technologies aims to achieve and how we will use them to achieve this. Beyond this, the chapter will show the mythology the team chose to use and why this was a natural choice to go with. Research in software testing methods will be included and the best fit for this project. Since this is a prototype project, a considerable time is put into this part of the project, to assure the team makes good choices when it comes to technologies to use.

2.1 Set to work with (another name perhaps)

Describes the existing stuff from twitter and twitter

2.1.1 Netflix

Dataset

somethingsomething

somethingsomething

2.1.2 Twitter

Twitter-API

somethingsomething

somethingsomething

2.2 Concept

Should maybe be in the intro?

The efficiency of the power user can always be increased. And one way of achieving this efficiency boost is through adding new functionality, but for bigger systems, this functionality adding can prove to be troublesome. Switching between the mouse and the keyboard can add up to be time-consuming in the long run, and also inefficient. With an environment where the user then instead can construct their own functionality, and keep their hands on the keyboard, can add tremendous value to the user experience. This can be made possible with a scripting environment, where the user themselves will be given the opportunity to work through a console on their graphical user interface. This can release them from the mouse and let them work more efficiently on the tasks at hand.

2.3 Similar solutions (Evaluation of State-of-the-art)

In this section, we will discuss similar solutions and their relevance to our project.

If this is not state of the art stuff, maybe separate this and that

Product	Counter-Strike
Concept	This game features a console that offers a wide variety of commands, including: - Changing the game options - Altering the game world - Player actions and cheats - Multiplayer communication and administration
Intended use	In a game, this functionality eases development and debugging, as well as increasing the modability and long-term value for players.
Similar products	Similar consoles also exist in other games, such as Carmageddon TDR 2000.
Relationship to our project	Like our project, the Counter-Strike console allows for using a DSL to work with objects in its given domain. It shows that the console can be a powerful tool that comes at a relatively small development cost compared to designing a GUI with a similar feature set.

Table 2.1: Counter-Strike Console

Figure 2.1: Counter-Strike Console

Product	Blender
Concept	This 3D morming software features a Python console with access to the morm data, animation data, etc. It is also common to extend the program using custom Python scripts.
Intended use	In a creative suite, this type of functionality can be used to perform operations that are not directly supported in the user interface. It is particularly useful for programmatically executing repetitive tasks that can be automatized.
Similar products	Similar solutions also exist in other creative tools, including a Python console in GIMP and Nyquist prompt in Audacity.
Relationship to our project	The Blender console exposes the underlying data structures to the user via an already widespread, powerful scripting language. This enables the users themselves to expand upon the functionality of the program and perform operations that would be prohibitively time-consuming to execute manually.

Table 2.2: Blender Console

Figure 2.2: Blender Console

Product	Firefox
Concept	This web browser features a Web Console where it is possible to execute JavaScript commands with access to the objects on the current page.
Intended use	In a web browser, this is useful for web development, prototyping and debugging for websites that use JavaScript.
Similar products	Similar features also exist in a few other browsers, such as Google Chrome.
Relationship to our project	Like our project, the Firefox Web Console offers access to the objects on a web page. It is possible for us to use the same principle, but for features that are useful for the end user and not just the developer.

Table 2.3: Firefox Web Console

Figure 2.3: Firefox Web console

Product	try.mongodb.org
Concept	This database website features a console where the user can execute commands on a dummy database.
Intended use	On this website, the console serves as an educational and demonstrational tool for people who don't want to invest too much time in learning about the database system.
Similar products	Comparable consoles also exist to allow the user to execute arbitrary queries in database administration tools such as PHPMyAdmin.
Relationship to our project	This console allows the user to execute queries directly on a database, and interestingly it allows input of objects with arbitrary structure. We will require persistent storage of our objects on a server, so this technology may be relevant.

Table 2.4: MongoDB Console

Figure 2.4: MongoDB Console

Product	web-console.org
Concept	This project provides shell access to a server through the browser window over HTTP with support for real-time communication.
Intended use	This system is intended to be used for server administration purposes when HTTP may be the only feasible method of connection.
Similar products	Similar solutions include access to unix Shells via VPN.
Relationship to our project	If, in our project, the server is designed to support user input using a DSL in a terminal window, then a solution like this one may provide the necessary functionality to embed a useful console into a Web UI.

Table 2.5: Web-Console

Figure 2.5: Web- Console

These products serve to illustrate that consoles are still applicable for purposes including software development, debugging, learning, extendability and where there is a high demand for flexibility. They can enhance productivity and provide features that would be prohibitively complex or expensive to implement in a graphical user interface. They also show that consoles do exist on the web, and that this is a field that is worth closer examination.

2.4 Development language and technologies (maybe not needed)

2.4.1 Database

For the system we need some kind of persistent storage, a database which all the clients can talk to to get the latest data in the system and to synchronize data from different clients. This database will be placed on a central server which all the clients can communicate with. For the database we are left with a decision between traditional SQL database or a so called NoSQL database. The differences are explained below.

NoSQL

NoSQL databases are a group of new emerging types of databases that are defined by the fact that they are addressing some of the following points: being non-relational, distributed, open-source and horizontally scalable [7]. NoSQL arose from the need to store large amount of data that do not necessarily follow the same schema, or share all the same attributes. One particular type of NoSQL databases that caught our attention early was the document NoSQL databases. The main characteristic of these databases is their schema- less structure. They also differ from SQL by generally not using a structured query language for data manipulation. They are easy to replicate and they offer simple APIs for the clients to communicate with. They are heavily customized for web- applications, and have gained much popularity in the modern web era. Most document NoSQL databases focus on quick replies to requests, as the query operation is by far the most common in a typical web- application. Because they typically are distributed, NoSQL databases are able to store enormous amounts of data, and they are often used within Big Data [13] applications like Hadoop [12], which recently has become a very popular subject within the computer science community. BASE [8] instead of ACID. Document NoSQL databases seemed like they would fit our project quite well. [6, 11]

SQL

The traditional SQL databases is by far the most common way of storing data in the world today. They store data in columns and tables, and add relationships between these tables. As a result they are referred to as relational databases. They focus on query optimization techniques and most of them use some kind of structured query language. Typically supports 4 basic operations which is select, update, rmete and insert. SQL databases are schema defined and follows the ACID(atomicity, consistency, isolation, durability) principles. [9]

Database Alternatives

Below, some of the database implementation available to us are discussed. We mainly investigated document NoSQL databases and regular SQL databases. The most appealing options are introduced below.

MongoDB

MongoDB is a large scale, high availability, robust system. It is a document NoSQL system, so instead of storing the data in tables as you would in MySQL, the data is stored in a document based fashion through for instance JSON with dynamic schemas. This makes the DB easier scalable horizontally. But the mongoDB still possesses the some of the great properties from relational databases, such as indexes, dynamic queries and updates. With mongoDB it is easy to map objects to programming language data types, which makes the DB easy to work with. The embedded documents and arrays makes the join operations less importance, which result in the ability to make reads and writes faster. JavaScripts can also be included in the queries. This makes mongoDB an efficient and high speed data base for storing objects and fetching to a system. MongoDB also has its own access console, where you can use scripting with Javascript language. [1]

CouchDB

CouchDB stores the data in JSON documents(NoSQL) object-oriented, which can be accessed through HTTP. The data documents can be transformed and data fetched with JavaScript. CouchDB has a lot of built in features which makes web development with CouchDB easier. It scales well through replication and is a consistent system, where CouchDB prioritizes the data of the user. CouchDB is multiversion concurrency control based, this is good for intense versioning, offline databases which resync later and

master to master replication. The interface to the database is REST based, which is a useful feature when you are developing a web- application. [4, 5]

MySQL

MySQL is the most popular database in the world of open databases. This is because of its high performance, reliability and ease of use. It should therefore be considered for the question of which database system to use. In opposition of the two database systems described above, MySQL is a relational database. This makes it more troublesome to work, with when it comes to JavaScript, than the other two. It is not as well integrated with JSON and will need parsing to be able to work with the clients. This alone is a hard negative towards MySQL. [3]

Conclusion

We decided to go for CouchDB in this project. We are working in a web domain, which CouchDB was designed for. While developing the console we will be working closely with JavaScript objects, and try to find ways of exposing these to the users. JavaScript objects are easily converted to JSON, the format used to store data in document NoSQL databases like MongoDB and CouchDB. As we only have the need to store the actual objects and a limited amount of relations between them, using CouchDB will ease our work considerably, and allow us to do things which would not be possible with a regular SQL database. CouchDB imposes far less restrictions on how you store your data than traditional SQL does. As long as the data is represented in JSON, you can store pretty much store anything you like, even within the same database. This will give us great flexibility when it comes to adding information to specific objects, and also means that objects of the same type can contain different attributes without us needing to create a new schema or change anything in the database. It also gives the user great flexibility in the sense that they can add any information they would like to the different objects, and we the developers don't have to plan for it at all, the database does all this for us. The fact that the customer suggested to us that we could use a NoSQL database, also tipped the scale in direction of the NoSQL alternatives.

Relational databases is the traditional way to deploy databases and they are in widespread use. In many situations they are extremely useful. However relational databases requires you to morm your data up front, before you save anything to the database. Failing to comply with these restrictions will lead to failure, and it is sometimes difficult to create this kind of morm that actually fits real world data. Even though objects may share common attributes, there are bound to be some attributes that are different. This is difficult to plan for in advance, and its the main reason we will not be using a SQL database for this project. For the database, we originally chose to use MongoDB as our document NoSQL database. This was due to the fact that it was better documented and seemed better suited for the system as we originally planned it. Some of the features CouchDB offered wasn't thought of as necessary for the project at the time. During the implementation process we however came to the conclusion that CouchDB actually was a better fit. This was mostly due to its ability to act as an standalone system on a server without the need for other supporting server technologies like Node.js or ASP.NET. Also, the fact that it automaticly creates a RESTful API to access the database turned out to save us a lot of time. As a result of these discoveries we changed to CouchDb during the third sprint.

2.4.2 Client-side web application technologies

Our main focus will be on the client part of the application, since this is the experimental part of our system, and it is this part that is visible to the user through the web browser. It is therefore important to choose a suitable technology for this.

Adobe Flash

A multimedia platform currently owned by Adobe. Is currently the industry standard for multimedia web applications. It excels at animation and 2D games, but its strong points are not likely to be useful in our project. A separate JavaScript is required to perform communication with a server and the development tools are costly.

Microsoft Silverlight

A rich media application framework developed by Microsoft. Useful for multimedia applications, but likely not beneficial for our project.

Java Applets

A technology that allows a Java AWT/Swing application to be displayed in a browser, backed by a Java Virtual Machine. Has excellent performance compared to other popular client side browser technologies. A signed applet can also communicate with a server using traditional sockets. It is possible to embed a scripting engine(for instance JavaScript). However, development effort may prove to become excessively heavy.

JavaScript

JavaScript is a scripting language supported by all popular web browsers. Has extensive frameworks built around it and allows for rapid development. It is also possible to let the user write commands using JavaScript directly.

Conclusion

As a team, we have extensive experience with Java, and less with the other technologies. However, we all have at least some experience with JavaScript, and we believe that it is the better choice for this project: The DSL can be implemented by allowing the user to perform operations on the JavaScript objects using (a subset of) the JavaScript language itself. There are also excellent tools for transfer and storage of JavaScript objects. Furthermore, communication between JavaScript and HTML elements is easily achieved.

JavaScript Related technologies

jQuery
A JavaScript library that simplifies how to use JavaScript to interact with the webpage, notably selection of Document Object Morm elements.

MooTools
A JavaScript framework that, notably, enhances the Document Object Morm and JavaScript's object oriented programming morm.

Dojo
A JavaScript toolkit offering asynchronous communication, a packaging system and systems for data storage. Intended to ease rapid JavaScript web development.

HTML5
A revision of the HTML standard currently still in development. Notably, it supplies support for multi-

media and more advanced user interface elements. Is commonly used in conjunction with JavaScript.

CSS
Cascading Style Sheets, used to specify a consistent look and feel to a series of HTML documents.

We decided to include jQuery in this project, coupled with HTML5 and CCS. jQuery was included because it is a technology we as a group have extensive experience with. It enabled us to easily select DOM elements, which is helpful when you want to create a dynamic web GUI.

2.4.3 Markup Languages

When communicating between the client and the server we need a data exchange format to represent objects and actions. The format has to be able to serialize and deserialize them on sending and receiving. The two alternatives most commonly in use today for solving this problem is XML (Extensive Markup Language) and JSON (JavaScript Object Notation).

XML

In widespread use in a lot of areas as of today and boasts great support and documentation. Originally meant to be a document markup language, but has over the years been used as a data representation language as well. It is suited to describe complex objects and documents, and it is easy to extend. Generally thought of as the more secure option of the two.

JSON

As the name suggest JSON is serialized JavaScript objects, well suited for web development, and very fast. JSON is easy for JavaScript to parse(we will be using JavaScript on both server and client), and the language has built in support for serializing and evaluating JSON data. Its small, simple, and easy to use. JSON is especially good at representing programming-language objects. It has gained a great deal of popularity in recent years, and it is well documented.

Conclusion

Both languages is well supported in almost all web related libraries, and they are both extensively documented. As security is not a main concern of this project, the fact that XML is more secure will not influence the decision. JSON will be used for this project. It covers the functionality we need, and it is generally thought of as the easier language to use. It is perfectly suited for web- development and JavaScript, which is the domain of this project. In addition the developers have more experience in using JSON than XML. JSON is also used to store objects in most document NoSQL databases, which we will be using for this project.

2.5 Software Testing

2.5.1 Testing Methods

The purpose of software testing is to uncover software bugs in the system and to document that the system meet the requirements and functionality that was agreed upon for the system. Testing can be implemented at any stage in the development process, traditionally it is performed after the requirements have been defined and the implementation is completed. In agile development processes however, the

testing is an ongoing process. The chosen development methodology will in most cases govern the type of testing implemented in a given project. [10]

Software testing methods are traditionally divided into white- and black- box testing. They differ mainly in how the test engineer derives test cases.

White- Box Testing

White- box testing focus on the internal structures of a system, and it uses this internal perspective to derive test cases. White- box testing is usually done at unit level, testing specific parts or components of the code. This kind of testing focus on how something is implemented and not necessarily why. Unit testing alone cannot verify the functionality of a piece of software is supposed to exhibit. It can uncover many errors or problems, but it might not detect unimplemented parts of the specification or missing requirements.

Black- Box Testing

Black- box testing handles the software as a black- box, meaning it observes the functionality the system exhibits and not the specifics on how it is implemented. The tester only needs to be aware of what the program is supposed to do, he doesn't need to know the specifics on how the functionality is implemented in the code. Black- box testing is typically performed to check if the functionality of the program is according to the agreed upon requirements, both functional and nonfunctional. Black- box testing is usually done at the component, system and release levels of testing. The main advantage of black- box testing is that no programming knowledge is needed to actually perform the tests. This way you can hire someone outside the development team who has had nothing to do with the implementation of the code to write and perform the tests, and you achieve as little ownership of the code as possible. An argument can be made though that this lack of insight in the specifics of the source code will result in repeated testing of some parts of the code, while other parts could be left untested.

Test Driven Development

The principle behind TDD is to develop the code incrementally, along with test for that increment. You don't move on until the code passes its test. The tests are to be written before you actually implement the new functionality. The process helps programmers clarify their ideas of what a code segment is actually supposed to do. The process is often used in agile development methods. Benefits from TDD include:

- Code coverage, every code segment should be covered at least one test.
- Regression testing, check to see if changes in the code have not introduced new bugs.
- Simplified debugging, when a test fails it should be obvious where the problem lies, no need for a debug tool.
- System documentation, the tests themselves act as a form of documentation that describe what the code should be doing.

[2]

Automated Tests

Automated offers the ability to automatically do regression tests, i.e. testing to uncover if any new code has broken a test that previously passed. If we opt for manual testing regression testing will be very time consuming as every test done so far has to be done over again. With an automated testing

framework this job will be a lot easier as you can run a great number of tests in a matter of seconds. Most development languages offers libraries for automated testing.

2.5.2 Testing Levels

Testing can be done at many different levels and in different stages in the development process. Following is the most common partitioning of testing levels and a description on each of them.

Unit Testing

Unit testing aims to check specific components, such as methods and objects. Typically you will be testing objects, and you should provide test coverage of all the features of that object. Its important to choose effective unit test cases, that reflect normal operation and they should show that the specific component works. Abnormal inputs should also be included to check if these are processed correctly.

Component Testing

Tests bigger components of the system, and their interfaces(communication with other components). Made up of several interacting objects. Component testing is mainly a tool to check if component interfaces behaves according to its specification.

System Testing

In a given development project there may be several reusable components that have been developed separately and COTS systems, that has to be integrated with newly developed components. The complete system composing of the different parts is tested at this level. Components developed by different team members or groups may also be integrated and tested at this stage.

Release Testing

Release testing is the process of testing a particular release of the system that is intended for use outside of the development team. Often a separate team that has not been involved in the development perform this testing. These kind of tests should focus on finding bugs in the complete system. The objective is to prove to the customer that the product is good enough. This kind of testing could either be based on the requirements of the system or on user scenarios.

User Testing

This is a stage in the testing process in which users or customers provide feedback and advice. This could be an informal process where end- users experiment with a new system too see if they like it and that it conforms to their specific needs. Testing on end- users is essential for achieving success in a software process as replicating the exact working environment the system will be used in is difficult to achieve during development. The end users can help provide feedback on how specific functionality will work in an actual work environment.

Another form of user testing involves the customer and its called acceptance testing. Its a process where the customer formally tests a system to decide whether or not it should be accepted, where acceptance implies that payment for the system should be made. Acceptance testing is performed after the release testing phase.

2.5.3 Conclusion

The concept of TDD is to develop exhaustive tests that specify the system, and then writing code with the goal of satisfying the tests. This is useful in systems where the key functionality is in the form of program logic that can be verified to conform to the specification. It is difficult to write such exhaustive tests in applications that rely heavily on GUIs, network connections and database systems because of the added complexity and heterogeneousness that these features involve. In addition, our prototype's exact specifications are likely to change during development, requiring large amounts of test rewriting in the case of TDD, because the tests will be more numerous and because the tests must be more strict. As a result we have opted not to use TDD in this project. We will however be utilizing unit tests with an automated testing framework where it is appropriate and will harvest some of the advantages linked to TDD, like the automated regression testing.

We will be writing unit tests throughout the implementation process and run these continuously. These tests will not be included in the report as test cases. The test cases will rather comprise of component and system tests. Component tests will be used to test specific components and their functionality as well as their interaction with other components. System tests will be used on the system as a whole to check if it meets the agreed upon requirements. These test cases will be derived from the requirements. We will also try include some acceptance tests to include the customer in the testing process.

We will be utilizing user testing and involve end users to get feedback on the entire system or specific functions. This testing will mainly be used to get feedback on the domain scripting language and how easy it is to understand and use. Preferably it will be done continuously throughout the development process. It is important to involve users as the goal of this project is to ease their workflow. As we are not working with an existing system thats actually in use, there will not exist any real users of this system with experience using it. We will therefore mainly be using our fellow students as test subjects, as they are readily available and technically competent enough to understand the concept and act as superusers. In addition the customer has stated that it will encourage employees from the entire company to us give feedback if we ask for it. Specific solutions can be sent to the customer representative which in turn will relay it to experts on the area it concerns.

Chapter 3

Requirements

Contents

How requirements were captured; discussion of major requirements (referring to Appendix A for details).

I "Requirement"-kapittelet forklar hvordan du satt opp kravene. Men ikke ha med en fullstendig oversikt over kravene her!

In this chapter, we will describe our initial product backlog and its rationale. The product backlog is in the form of a list of user stories describing the overall requirements for the system. Note that because of the product's nature, this list would be subject to extensive changes in later sprints; It is merely a starting point.

3.1 Choice of Domain

Our project's problem is a rather general one, not intrinsically tied to any specific domain. However, its existence depends on an actual area of application. For this reason, it has been necessary for us to discover a domain for which to implement our prototype. Our criteria for selecting such a domain were as follows:

- It should be simple enough for test subjects to understand and feel familiar with.
- It should be complicated enough to demonstrate the problem.
- The console has to be useful in the domain, in such a way that it eases the workflow or opens new possibilities
- Object oriented design should be applicable
- There should be potential for the existence of power users capable of using the console

Various domains were considered, including but not limited to banking, warehouses, project management, travel agencies, education, social networking, music management and health care. Ultimately, based on the criteria listed, we decided in consensus to implement our solution for a library system.

3.2 Use cases

As discussed earlier, our domain is a library system. This leaves us with objects such as books, authors, employees and customers. Possible use cases would include:

- Register a new customer
- Register a new employee
- Order a new book
- Add a new book to the system
- List books in the system
- Search for books
- Place a reservation on a book
- Record a borrowing of a book
- Record a return of a book
- Extend a borrowing of a book
- Register the state of a returned book
- Remove a book from the library
- Request that a new book is ordered
- Export inventory
- Find the location of a book
- Generate reports

Figure 3.1: Use Case Diagram - Customer

Figure 3.2: Use Case Diagram - Employee

3.3 User stories

User stories are sentences describing requirements of users and justification of those requirements. They are written in the perspective of the end user. Our customer was unable to supply us with an explicit set of such user stories, so we composed a list of candidate user stories from our common understanding of the system gained through earlier meetings with the customer. The user stories are presented from a few different points of view: The administrator section describes detailed technical requirements. The general section contains stories applicable on general problems. The domain specific section contains user stories that are specific to the library domain. [1]

We proceeded to collectively estimate the difficulty of implementing each user story, as presented in the list below. The list of user stories was reviewed by the customer and approved as an initial product backlog. We would use this list for prioritizing which functionality to implement during the planning meeting for the first sprint.

Administrator point of view

A1 As an administrator, I want to be able to store objects in a persistent database, so that I can migrate data easily. Difficulty level: 5.

A2 As an administrator, I want to reflect the changes in persistent storage back to user sessions within one second, so the users always see the latest updated data. Difficulty level: 6.

A3 As an administrator, I want to be able to work directly with object attributes rather than any form of raw data. Difficulty level: 4.

User point of view - General

G1 As a user, I want my saved actions to be replicated on to the server within one second, so that my actions will not be lost and the client and server is consistent. Difficulty level: 3.5.

G2 As a user, I want my changes to be propagated to other users of the system real- time. Difficulty level: 2.

G3 As a user, I want to be able to see the console and graphical interface at the same time, so that I can use them both side by side simultaneously. Difficulty level: 8.

G4 As a user, I want the changes in console reflect in graphical user interface and likewise, so that I can have overview of the changes I made and I can understand easily, how the system works. In addition this will ensure consistency between the console and graphical interface. Difficulty level: 4.

G5 As a user, I want access to a tutorial, so that I can learn to work with the system easily. Difficulty level: 3.

G6 As a user, I want to be able to display the currently available commands in the console, so I can easily see what I can do with the objects at hand. Difficulty level: 3.

[1] http://scrummethodology.com/scrum-user-stories/

G7 As a user, I want to be able to easily repeat and edit last command, so that I can use it on another object. Difficulty level: 2.

G8 As a user, I want to be able to use batch commands, so that I can work with more than one object at the same time. Difficulty level: 3.

User point of view - Domain specific

D1 As a user, I want to be able to add a new book to the system using both the console and graphical interface. Difficulty level: 1 (reference).

D2 As a user, I want to be able to rmete a book in the system using both the console and graphical interface. Difficulty level: 0.5.

D3 As a user, I want to be able to edit information on a specific book and save these changes using both the console and graphical interface. Difficulty level: 2.

D4 As a user, I want to be able to search for a specific book in the system, so that I can watch the information on it using both the console and graphical interface. Difficulty level: 0.75.

D5 As a user, I want to be able to list all the books currently in the system, so that I easily can get an overview of all the books currently in the library. This should be possible in both the console and the graphical interface. Difficulty level: 1.5.

D6 As a user, I want to be able to register a new customer in the system, so that customers can be saved in the system. This should be possible in both the console and the graphical interface. Difficulty level: 1.

D7 As a user, I want to be able to register when a customer borrows a book, so that the information is stored in the system. This should be possible in both the console and the graphical interface. Difficulty level: 1.75.

D8 As a user, I want to be able to edit the information on a specific borrowing of a book, so that any changes can be recorded. This should be possible in both the console and the graphical interface. Difficulty level: 2.5.

D9 As a user, I want to be able to list all the books currently borrowed, so that I can get information on each of them. This should be possible in both the console and the graphical interface. Difficulty level: 2.

D10 As a user, I want to be able to reserve a book for a customer, so that customers can request and reserve certain books. This should be possible in both the console and the graphical interface. Difficulty level: 3.

D11 As a user, I want to be able to list all the reservations currently in the system. This should be possible in both the console and the graphical interface. Difficulty level: 2.

D12 As a user, I want to be able to view reservations on specific books, so that I can see if a specific book is available for borrowing at the moment. This should be possible in both the console and the graphical interface. Difficulty level: 1.5.

D13 As a user, I want to be able to order new books for the library using both the console and the graphical interface. Difficulty level: 1.5.

3.4 Summary

The envisioned system is a web application with an embedded console, see. The user will be able to choose if they want to use the console or the more traditional web user interface.

In the system, users will be able to manipulate a list of books by adding, removing and editing books, and perform filtering and searching. In addition, the system will have a list of customers, who are able to place reservations on books. Books can also be borrowed by customers.

Bibliography

[1] 10gen Inc. MongoDB Introduction. "http://www.mongodb.org/display/DOCS/Introduction", 2012. [Online; accessed 2012-10-12].

[2] Kent Beck. *Test-driven development : by example*. Addison-Wesley, Boston, 2003.

[3] Oracle Corporation. MySQL About. "http://www.mysql.com/about/", 2012. [Online; accessed 2012-10-12].

[4] The Apache Software Foundation. CouchDB about. "http://couchdb.apache.org/", 2012. [Online; accessed 2012-10-12].

[5] The Apache Software Foundation. CouchDB technical overview. "http://wiki.apache.org/couchdb/Technical%20Overview", 2012. [Online; accessed 2012-10-12].

[6] Neil Leavitt. Will nosql databases live up to their promise?

[7] NoSQL. List of NoSQL databases. "http://nosql-database.org", 2012. [Online; accessed 2012-10-10].

[8] Dan Pritchett. Base: An acid alternative. *Queue*, 6(3):48–55, May 2008.

[9] Raghu Ramakrishnan. *Database management systems*. McGraw-Hill, Boston, 2003.

[10] Ian Sommerville. *Software engineering*. Pearson, Boston, 2011.

[11] Cristof Strauch. Nosql databases.

[12] The Apache Software Foundation. Welcome to Apache Hadoop! "http://hadoop.apache.org/index.pdf", 2012. [Online; accessed 2012-11-10].

[13] Wikipedia. Big data. "http://en.wikipedia.org/wiki/Big_data", 2012. [Online; accessed 2012-11-10].