Table of Contents

Project Establishment

Background and Purpose

Background for this project is to develop a stock-screener, which is like a search engine system for an on-line shares community website: www.freyr.dk

This final project is implemented after a thorough study and understanding the concepts of project management, system development methods, database, programming and web services which I learned in earlier semesters. By doing this project, I hope to get better practical knowledge and get real world job experience.

This Project has to cover:

1). Create a "crawler" to extract information from the websites and save it into the database regularly.

2). Create a web front-end for the database, where users can search for the shares (the screener).

Goals

I am defining here **SMART** goals which I wish to reach in this 10 weeks project.

1). I have to complete the project establishment, requirement specifications together with customer within 2 weeks and selecting process model.

2). Every 2nd.week I will track differences between estimated time and real time phase to control delays, by updating the time plan.

3). I have to be able to complete three XP iterations before end of project.

4). Report and product has to be submit to the school before deadline that is 31.Oct.2008 at 12:00pm.

Requirement Specifications

I made few user stories together with customer and wrote down here as customer accepted functional requirements for system. These requirements could be changed or canceled or added new requirements by customer further. Working papers for user stories are found in the **Appendix-G**

Uno.	Requirement	Description	Estimate time	Actual time spent
1	Admin interface to manage criteria	Criteria is defined by the four fields: Name, URL, XML path, Convert Expression.	1 day	1 day
2	Import information from Reuters	Shares information has to extracted from Reuters website and added to database, including fields PE/Ratio, shares_out and Div&Yield.	2 days	3.6 days
3	Distribution graph for each criteria	Distribution graph helps to find good values for criteria, So Freyr prefer to have a graph for it.	2 days	2.3 days
4	Import should run daily for all symbols	There is a list of all symbols in a Frey's database . All those symbols have to be updated everyday.	3days	2 days
5	API : Get information for symbol.	There has to be an API , where it is possible to get all known data has to able to search.	2days	skiped
6	Show a Search form	When user enter into application, it has to display a search form which has fields such as Criteria, Market Cap, PE/Ratio, Dividend Yield, 52w Price change etc.,	3days	4.3 days
7	Show Search results	When user enters Criteria and clicks on Search button then related search form result has to be shown. Company name and symbol has to link to details page when user clicks on them.	2days	1.2 day
8	History of extracted values	Extracted values should be saved also for history purpose	3 days	3 days

Risk Analysis

By Risk analysis, we improve the accuracy of traditional scheduling, cost estimation and systems engineering skills. Risk is a combination of the severity and probability of the hazard event. This means that by analyzing and categorizing the risks, I get a better understanding for the risks that can appear, and by the better understanding I have more chance to avoid them.

In the table below, are the risks I identified, their probability, the effect on my project, and how accepted this risk is. Together with this are some solutions on how I can avoid the risks.

Risk	Probability	Effect	Acceptability	Risk Solutions
Requirements are wrong.	Low	Medium	Acceptable	Find the missing requirements in next iterations.
System is not user friendly.	Low	High	Intolerable	Test together with customer to improve.
Software is not good enough for safety.	Low	High	Intolerable	Improve safety requirements in further iterations.
Code and report is lost.	Low	High	Intolerable	Take regular backup to Freyr's server.
Not well Documented	Medium	High	Intolerable	Make use of weekly meetings with supervisor to get input on what to improve.
Delay of project report submitting.	Medium	High	Acceptable	I fail the semester, But, I guess there are 2 more chances to finish. Freyr can use the well-developed parts for further development.
Working with new technologies. Not sure that technologies can solve the problem.	High	High	Acceptable	Work with Freyr to gain better knowledge, document if something can not be solved.

By those risks and their solutions I found are suitable to use XP process model to minimize the risks and saves the cost of project.

Strategy Analysis

Strategy analysis is a way to find strategic elements to include in the project. By looking at the conditions of the project, and the strengths and weaknesses of those, it is possible to find their strategic solutions. Like with risk analysis, it is a way to get a better understanding of the project, to anticipate future problems with more ease. Below are the strategic conditions I have found for the project:

Conditions	Strengths	Weaknesses	Strategic element
Technical Issues	Good knowledge on system development methods, programming and SQL server. Some knowledge on Python, framework and My SQL.	Don't have experience with Django at all.	Self study and finding information on Internet. Help from Freyr developers about introduction.
Developer(s)	Only my self. Getting help with pair programming from my husband Søren on Python and Django.	Only myself working on project report, analysis and design phases and missing team work inspiration.	Confirm project steps with supervisor. Pair programming and Short meetings with customer. Testing and review each iteration to improve quality.
Result	Clear defined goals.	Difficult to estimate , due to experimenting on new subjects and complexity.	Use XP with many short iterations.
Users	Small time and private investors after release the product.	No real users at developing phase.	Work more close with Freyr. show prototype to customer to try.
Environment	No technical problems, no disturbance, good cooperation from Freyr. peaceful environment at home.	Faraway from home to school to seek help from teacher.	Internet available at home, so I can discuss about project with Freyr on line.

Strategy: Following these strategic elements, I am going to use agile method with 3 iterations and prototypes by using XP process model.

Selecting process model

By the risk and strategy analysis, I have selected to work with the extreme programming (XP) process model.

XP is one of the agile process models. To be agile means to move quickly, and light weight - this sounds like a good features that I can use in this project.

- Waterfall. I used this for the first 2 semesters and also I used Iterative process model in 3rd and 4th semester, So I am curious to try with XP in practical and get better knowledge, what I learned theoretical in system development methods.

- There is more uncertainty in the project, and agile is a way to handle that.

- XP allows to do continue releases, so customer can start use them before completion of project.

- Since I have very little knowledge on Python and Django, it is good opportunity to gain knowledge by working on pair programming together with Frey's developer.

- By using XP, I can reduce the cost of changes in project.

For these reasons, I would like to select an XP process model, since it allows me to produce more than one iteration, and to put more focus on what is the important parts to get done first, and this is then XP.

Below is this diagram showing the phases of XP process model life cycle, seen from the iteration (meaning there will be more iterations, each iteration looks like this).
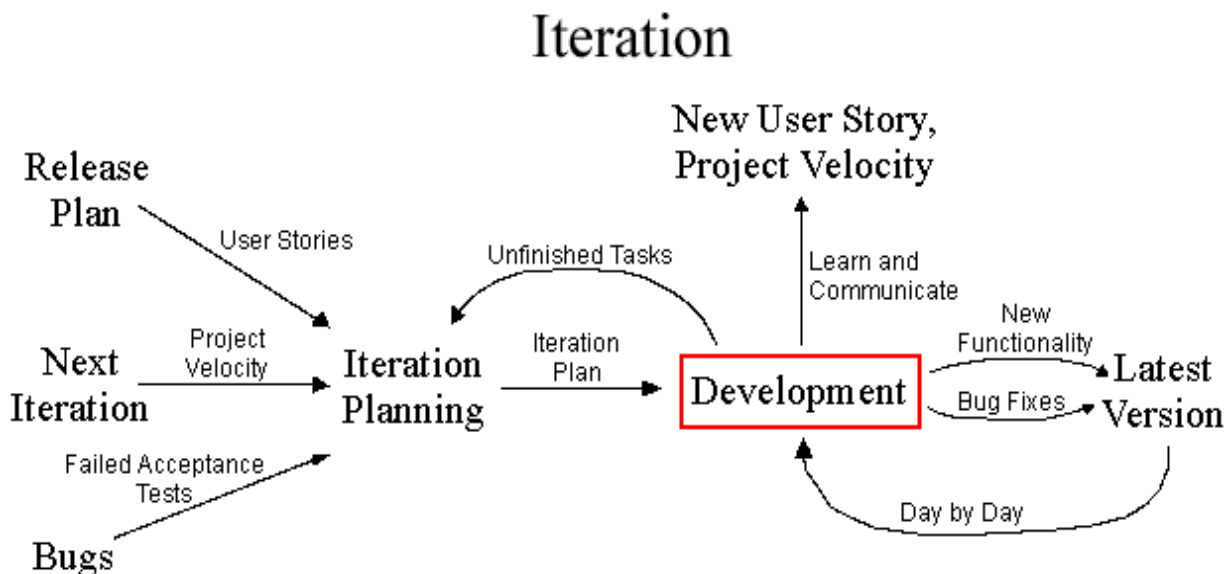
Initial -Time planning

As a final point on my project establishment, an initial time-plan is good. Since I am only one person on the project, and since the estimates of what must be done this early in the project is very uncertain, the time-plan will only be rough estimates of the project milestones. The milestones are defined by the artifacts they produce, so it is clear when a milestone has been reached.

Title	Description	Artifacts	Time estimate	Week
Project establishment	Discussion with Freyr about the project requirements, write about process model selection, create initial time plan.	Project contract and initial time plan.	1 week	34
Release planning	Find user stories for the project, give user stories to customer to select for first iteration.	List of minimum 10 user stories.	1 week	35
Introduction to Freyr	Get introduction to the Freyr setup, look at Python, Django, and MySQL.	Document describing how is C# vs Python, ASP.NET vs Django, and MySQL vs MSSQL.	1 week	36
Iteration planning #1	Agree on user stories to implement in first iteration.	List of selected user stories.	1/2 day	36
Iteration development #1	Work on the first iteration. (My guess is database related tasks)	Detailed design, tests, code, and test report for selected user stories.	2 weeks -1 day	37+38
Iteration planning #2	Review the produced artifacts. Agree on user stories to implement in second iteration.	Review document. List of selected user stories, and/or error reports.	1 day	38
Iteration development #2	Work on the second iteration. (My guess is crawler related tasks)	Detailed design, tests, code, and test reports for selected user stories.	2 week -1 day	39+40
Iteration planning #3	Review the produced artifacts, Agree on user stories to implement in third iteration.	Review document. List of selected user stories, and/or error reports.	1 day	40
Iteration development #3	Work on third iteration. (My guess is web GUI)	Detailed design, tests, code, and test reports for selected user stories.	1 week -1 day	41
Review of iteration #3	Review of the produced artifacts. Evaluate cooperation with Freyr.	Review document.	1 day	41
Report writing.	Finish the report.	Final report.	1 week	43

Time plan review

By this Time plan review, I realize that at third iteration I can not implement rest of the all user stories, Freyr would like to get done with 8th user story which is more important to them than 5th.So I just decided to give up the 5th user story which is not so important. My supervisor Jens also suggested me to focus on report writing on last week. I am happy that I successes to implement rest of all seven user stories.

Title	Est_time in weeks	Actual_time spent in weeks
Project establishment	34	34 (Started on 21-August-2008)
Release planning	35	35
Introduction to Freyr	36	36
Iteration Planning #1	36	36
1.st. Iteration Development.	37 +38	37 + 38
2.nd. Iteration planning & development	38	38
2.nd. Iteration Development	39 + 40	39 + 40
3.rd Iteration Planning and development	40	40
3.rd Iteration Development	41	41 + 43
Review of all project, report writing and submitting to school and Freyr.	44	44

I can see that in general the time plan was followed, except in the end , where I became behind schedule. I think this was because I got stress by working on report and code at the same time.

Cost Estimation

I am using **Agile COCOMO II** tool to estimate cost and effort for this project. It is a simple method of software cost estimation by analogy to a previous project. It works by using the lessons learned in previous projects, and by comparing cost/effort "drivers" with what I know about the current project.

I used the cost drivers for which I had knowledge enough to estimate, compared with my previous project SIM-ATC. The estimated effort is 3 months, which is roughly the time of this project. The estimation came up with 2.81 man months for the full project, which I hope is realistic. The full details for the cost estimation output can be found in **Appendix-H**.

Programming language and tools

In my previous projects I have used C#, .Net, and ASP.NET for implementation, and other Microsoft tools such as MSSQL server and Visual Studio .NET. I have some kind of familiarity with these tools, but, it is not a requirement that I must use those for this project. I agreed with Freyr to work with the tools such as Python, Django framework and Mysql which Freyr already using. Because I am also curious to learn new technologies by practical. So using new tools is a way to show that I am capable of applying learned theory regardless of the tools.

- Django framework is presented as a " high-level Python web framework that encourages rapid development and pragmatic design..", which is good fit for an agile process.
- Python, Django, and MySQL are already in use by Freyr.
- All are available both for Windows platform which I am familiar with, and for Freyr.dk's selected platform.
- They are new technologies, but not radical new, so they should be able to learn in the project time frame.

Comparing with known technology

In order to assess these new tools, I will create a small comparison. This comparison is mostly for my own use to see the differences, but also to document what I have learned during the project.

Python and C#

Python and C# are both object oriented programming languages, and they are both procedural, but from that they are also some different. I will try to give a short overview of how is Python different from C# in relation to the features I am using in my project.

I think it is easy to get started with Python. The basic structure is almost like C#, but just little different. Instead of curly-brackets, Python is using indenting to decide the scopes in the code. Also I think it is more object oriented, everything in Python is objects, also methods and documentation in the code. It has types, but it is more dynamic, so I do not need to bother about declaring types on variables.

I show below here a small example program I made while starting with Python, and a C# program with same features, to give idea of basic Python structure:

```
C# program to calculate fibonacci numbers normal and recursive.
using System;

namespace simple {

    public class Fib1 {
        // iterative
        public int Calc(int HowMany) {
            int a = 1, b = 1;
            for (int i = 1; i < HowMany; i++) {
                int tmp = a;
                a = b;
                b = tmp + b;
            }
```

```
            return a;
        }
    }

    public class Simple {
        // recursive fibonacci
        public static int Fib2(int HowMany) {
            return Fib2(HowMany, 1);
        }
        public static int Fib2(int HowMany, int a) {
            if (HowMany > 1) {
                return a + Fib2(HowMany - 1, a + 1);
            }
            else {
                return a;
            }
        }
        public static void Main(string[] args) {
            Console.WriteLine("by class: " + new Fib1().Calc(10));
            Console.WriteLine("by recursive: " + Fib2(10));
        }
    }
}
```

There is not any news here, I created a static method and an object method, and both are working to calculate Fibonacci, one does it iterative, and another recursive.

Python program doing the same

```
# class to calculate fibonacci
class fib1:
    # method to calculate
    def calc(self, how_many):
        a, b = 1, 1
        for i in range(1, how_many):
            a, b = b, a + b
        return a

# method to calc fibonacci recursive
def fib2(how_many, a=1):
    if how_many > 1:
        return a + fib2(how_many-1, a + 1)
    else:
        return a

print "by class: ", fib1().calc(10)
print "by recursive: ", fib2(how_many=10)
```

We can see here the differences now:

- The program is much smaller, exactly half size if we count the "using" statement in C#. I think smaller program means more easy to read and understand.

- There is less unnecessary syntax. The C# example is full of () and {}s. This also makes it easier to understand.

- Indentation in Python takes some time to get used to. It is annoying that I have to be so precise.

- Python is using files for modules, it is the same as "name space" in C#, but it seems more

simple that one file is one name space.

- There are no type declarations for variables in the Python. This makes it easier to read, but also harder to know what to pass if in doubt.
- "if", "for", and structures in general are similar to C# and Python, so it is easy to understand concepts of both.

Static or dynamic language

It is not necessary to compile Python programs, the Python virtual machine (PVM) will do that automatically when program is loading. C# is also running in a virtual machine Common Language Runtime (CLR), so in that way they are similar.

Static language is to my understanding that the program can be evaluated at compile time, and not while it is running. The opposite is then dynamic, it evaluates when it is running, meaning some errors you can only see while program runs, also simple errors like for example spelling mistakes of method names, those are shown during compile in C#.

But, dynamic language like Python also offer me some functionality that I can not get from C#, which I used during my Extractor module. It is using eval() function, which evaluates a piece of code at runtime, that way I can put python code into the database, load it, and have it evaluated when I need it.

IDE support

For my project, I used the Python IDLE.IDE . It is a very simple IDE, almost like a text editor, that can do syntax coloring, and helping with the indentation.

Compared to Visual Studio, this is very basic. Visual Studio has many features besides writing code, such as managing all files of a project, browsing structures such as name spaces, classes, and methods, and also it has a whole set of features for the GUI construction.

One thing I am tired with in Django is, I have to be memorize syntax for commands at console. While working with Visual Studio, it made me feel more comfortable by using GUI execution in earlier projects. I think Python/Django should add GUI integration.

Django and ASP.NET

I chose to compare Django with ASP.NET, since both are web application frameworks. There are some parts of Django which are not part of ASP.NET but other .NET frameworks – in those places I will mention that.

Django is a web application framework, which is using Python as the programming language, just like ASP.NET is a web application framework, which is using C# as its programming language. It is based on the MVC pattern, but they use some different words than MVC.

I will note in this chapter what I see as the proper MVC components. It is hard to say if ASP.NET is a MVC framework or not – you can chose to say it is, where the Code-behind is the controller, the ASP page is the view, and the model is your database, or, you can say that it is not, because there is no clear distinction between controller and view, and there is no model.

Models

One part of Django is an object relational mapper (ORM), that means some framework for mapping objects to and from a relational database. Django supports more databases, and it is very easy to configure which database you want to use, by the application settings file. For development it is possible to use a file-based database, SQLite, which is like Microsoft Access, and does not require any server installed.

The models is where to program the business objects for use by the application. Each class has defined attributes which specify how to map to database, and it must inherit from Django class models.Model. In my example below I am showing how one model can look like:

Models.py file from search application

```python
from django.db import models

# The business object Stock Property
class StockProperty(models.Model):

        # Define the attributes and their mapping to database
        name = models.CharField(max_length=100)
        url = models.URLField()
        xml_path = models.CharField(max_length=250)
        convert_expression = models.CharField(max_length=250, null=True, blank=True)
        frontpage = models.BinaryField()

        # string convert method
        def __str__(self):
                return "%s url=%s, xml_path=%s, convert_expression=%s" % \
                        (self.name, self.url, self.xml_path, self.convert_expression)
```

By this model, I can now work with it in the rest of the application through the API that is provided for all models. Below I show some examples of how I am working with this model.

Views example working with models API

```python
# Views for search application
from django.shortcuts import render_to_response

def index(request):
        """ Index shows the a default criteria """
        # find all StockProperty which have frontpage=True
        list = StockProperty.objects.filter(frontpage=True)
        return render_to_response("index.html", {"properties": list})

def all(request):
        """ Show all criteria """
        list = StockProperty.objects.all()
        return render_to_response("index.html", {"properties": list})

def change_name(request, the_id, new_name):
        """ Change the name of one criteria """
        # load from database
        p = StockProperty.objects.get(id=the_id)
        # modify name
        p.name = new_name
        # save to database again
        p.save()
```

```
def create(request, the_name, the_url, the_path):
        """ Create a new criteria """
        p = StockProperty(name=the_name, url=the_url, xml_path=the_path)
        p.save()
```

So, it is very simple to work with the business objects in Django, it will handle all the SQL for me. By using a database client, I can go to the database and verify that all is done correctly.

ASP.NET does not have any ORM features like this that I know of. They have ADO.NET, but to my understanding, that works in a different way, with "generic" containers (like GridView), that maps to the database, and you have to create the SQL for updates, inserts and deletes. I think working with Django is more easy here, because it is easier to relate to the business objects that you write yourself.

Dynamic Admin Interface

One other thing I used in Django was its Dynamic Admin Interface (DAI). The DAI is a full web application that Django creates by looking at the models I have defined. It creates a GUI where I can perform CRUD operations on all the models. This is very good, for example for configuring the application.
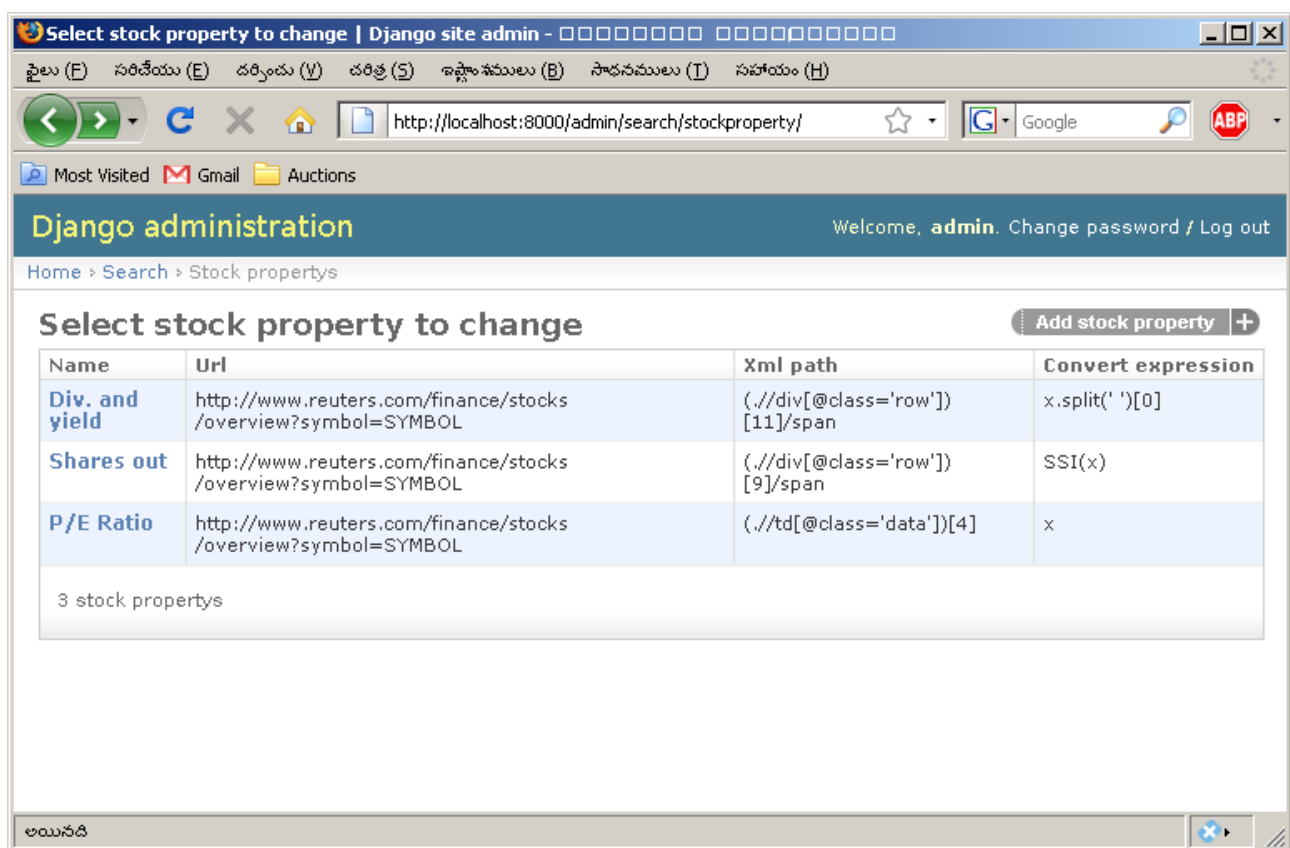
Admin interface where it is showing a list of tuples in one of the tables (StockProperty)

Admin interface where I am adding a new tuple (Stock Property)

It is possible to configure how the models are presented in the admin interface, for example I have an admin configuration like the following, where I define which fields of StockProperty that I want to show in list.

Admin.py showing admin interface customization

```
from models import StockProperty
from django.contrib import admin

class StockPropertyAdmin(admin.ModelAdmin):
    list_display = ['name', 'url', 'xml_path', 'convert_expression']

admin.site.register(StockProperty, StockPropertyAdmin)
```

I think it is very simple to work with the models using the DAI. I do not need to work with SQL at all to create configuration in the database, I can just use the DAI GUI, which is much easier than SQL :-).

This is something that ASP.NET does not have at all, there are maybe some tools you can buy to get this, but it is outside the scope of my project to look at that.

Design URLs

In ASP.NET, one .aspx file is the same as one URL.

With Django I must define all URLs, and what views should execute when an URL is requested by

the client's browser. Below I show one example of how I define the URLs for one of my applications in the project.

Urls.py defining urls to views map.

```
from django.conf.urls.defaults import *
from search.views import do_query, show_criteria

from django.contrib import admin
admin.autodiscover()

urlpatterns = patterns('',
    (r'^search/', do_query),
    (r'^show/(?P<criteria>\d+)', show_criteria),
    (r'^admin/(.*)', admin.site.root),
)
```

I define the search/ url to go to a query view, and then a show/ID to send the ID to a show_criteria view. It is not so easy to work with those parameters, so I do not use that a lot. It is using regular expressions, which are quite hard to understand.

It is hard to say what is better, but I think the Django model offers more flexibility, while the ASP.NET way is just simple. I like with Django that it is possible to get parameters from the URL directly.

- ASP.NET uses filenames to specify the URLs. Values are passed by normal request encoding.

- Django is allowing to design URLs and also to pass values from the URL to the view directly.

Design templates

To design the GUI is completely different in Django and ASP.NET. The main differences can be listed as below:

- ASP.NET uses components that renders HTML. By configuration of components it is possible to get different output.

- Django you have to "program" HTML, in the HTML you can add content by variables.

Both ASP.NET and Django have some kind of separation between the surrounding layout and the page contents. In ASP.NET it is called Master Pages, in Django it is modeled as programming where I "extend" a template (inheritance), and replace blocks.

I think it is easier with the ASP.NET to create a GUI, especially because I can preview it inside Visual Studio, but it is less flexible. If I need to create a GUI where the available components are not enough, then Django is probably the only choice.

Custom admin commands

Django-admin.py is a tool to administer a django application, for example, to create a project, to start a development server, etc. It can also be extended with your own admin commands. I am thinking to use this for solving periodical jobs (eg. It runs from commandline, but inside the django framework).

Example command for the "importone" command.

```
from django.core.management.base import LabelCommand
```

```
class Command(LabelCommand):

    help = 'Import all stock properties for one symbol.'
    args = '[symbol]'
    label = 'symbol'

    requires_model_validation = True

    def handle_label(self, symbol, directory=None, **options):

        from django.conf import settings
        from stockscreener.search.models import StockProperty
        from stockscreener.search.extractor import Extractor

        stock_properties = StockProperty.objects.all()

        for stock_property in stock_properties:
            print "Extracting %s for %s" % (stock_property.name, symbol)
            # create instance of Extractor
            extractor = Extractor()
            # call method extract on the instance of Extractor
            extractor.extract(symbol, stock_property)
```

The commands inherit from the Django object Command, in this case I am inheriting from a specialized Command called LabelCommand, which handles commands that requires one argument, which is what I need. The documentation on all this is not so good in Django, and I had to ask help to find out where to look at examples on how to create these commands, but Freyr is using this so they were helpful with it.

MSSQL and MySQL

For my project I chose to use MySQL. This was partly a requirement from Freyr (they only want to use OpenSource tools), and, because it is one of the supported databases by Django.
In my previous projects, I have only used MSSQL Server 2005, so also this is different from my known. In this chapter I will try to discuss my impressions on how MySQL is different in working from MSSQL Server.

I installed both MSSQL and MySQL so I can compare better. When I have to download MSSQL Server, I see that there are two different versions, one "express" version which is free, but has limits on the size of the database, and one other which is only available as 30 days try before buy, but has no limits on use. For my project is clear that Freyr would not be able to use the free version since the database will grow over time, so I compare with the 30 days test version.

IDE support and features

To get started with MySQL is as easy as MSSQL Server. It is very small in download size compared to MSSQL Server (100MB compared to 900MB), but during installation I can also see that it has not so many advanced features as MSSQL Server. MSSQL Server has advanced features like Integration services, and some Business Intelligence analysis (I don't know what this is for, but I did not select it during installation). These features are not available in MySQL. But, for developers, I think it is good if MSSQL Server was available in a small version that does not take so long time to download.

The tools I get with MySQL are very good considering they are free. For MSSQL Server I get one tool SQL Server Management Studio, where I can do all work inside, but for MySQL I get five different programs that cover almost the same as the Management Studio does. There is an very advanced database design tool, where I can work with a graphical database design, I could not find such a functionality in the SQL Server Management Studio. However, in my project, I did not need this functionality, so there is no difference here.
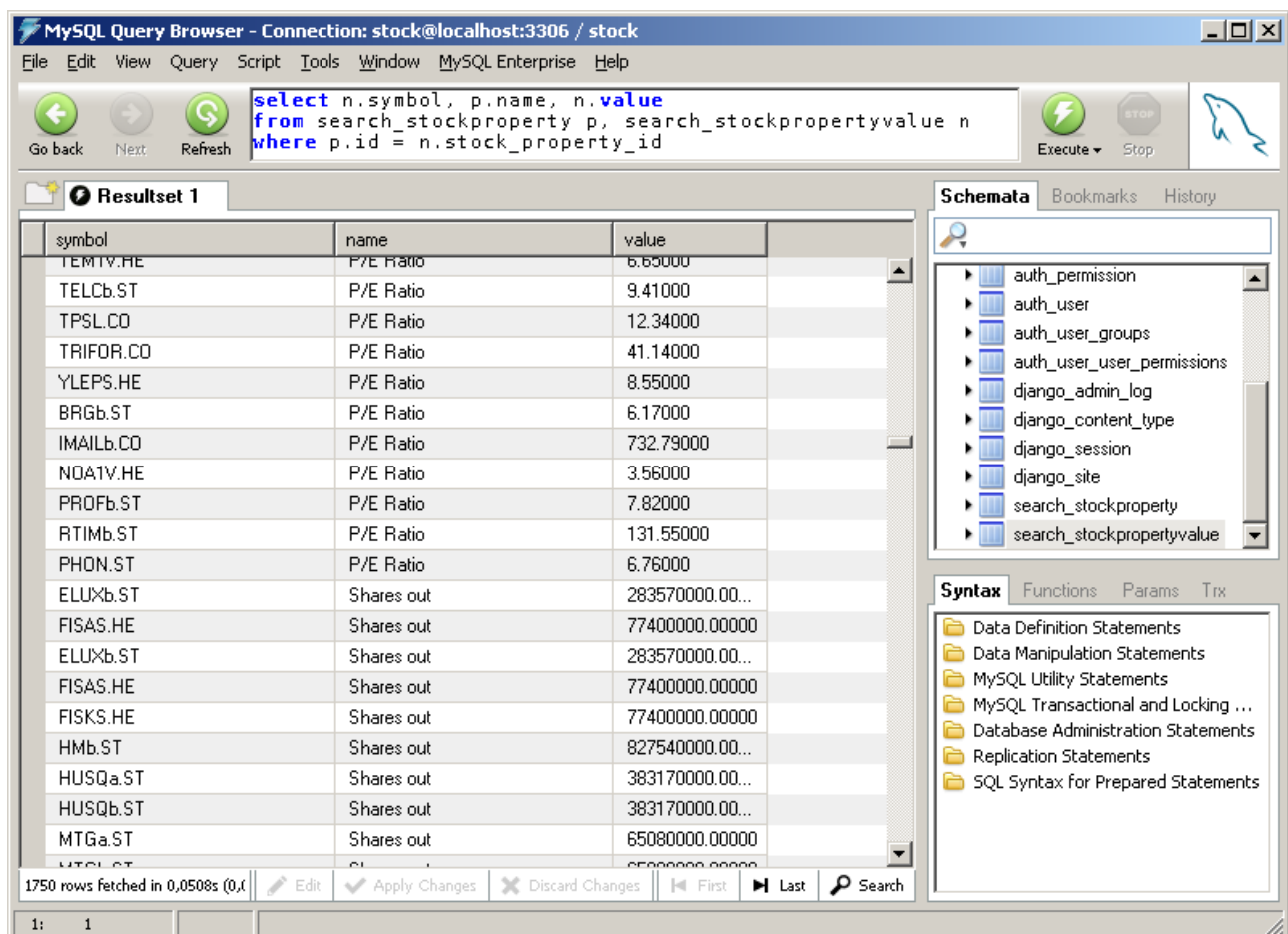
A sample query from the MySQL Query Browser (the tool for querying and working with DDL). It is a very simple interface where it is easy to create queries (in SQL Server Management Studio I have to create new Query and then enter query), but also there is rich help for building queries.

Storage engines

While MSSQL is always working in the same way, MySQL has different ways of storing the data "physically". This they call storage engines. There are many different storage engines, and each of them is fit for some specific situation. The most common ones are:

- InnoDB: This is a storage engine using file-system and supports transactions.

- MyISAM: This uses file-system, does not support transactions, but it is very fast.

When a table is created, it is possible to specify the storage engine, see the example here below:

Creating two tables with different storage engines

```
create table stocksymbols (
   id int primary key,
   name varchar(200)
) engine=MyISAM;

create table stocktrades (
   id int primary key,
   symbol_id int not null,
   value decimal(10,4)
) engine=InnoDB;
```

I think it is a good idea by MySQL. With some knowledge about the engines, it is possible to optimize the database performance by selecting a proper engine. I don't think that MSSQL Server has the same features, but it has other ways of optimizing tables by parameters.

My application will not require transactions, so I will use the default MyISAM, for speed.

SQL

Both databases supports all the standard commands such as SELECT, INSERT, UPDATE, DELETE. From what I can read they both conform to the standard SQL-92, but I am not sure what is exactly in this standard. It should mean though, that most of the SQL is the same on each of the databases.

When I am running test queries, I have not encountered anything that does not work on one of them. There is some difference with how automatic Ids is working, as an example I am showing here below.

Create table, insert, and get new ID, MSSQL	Create table, insert, and get new ID, MySQL
```create table stock (id int identity(0, 1) primary key, name varchar(100)); insert into stock (name) values ('KST.IT'); select @@IDENTITY; > 0```	```create table stock (id int auto_increment primary key, name varchar(100)); insert into stock (name) values ('KST.IT'); select last_insert_id() > 0```

As we can see, little different, but in general the same concept.

Programming

MSSQL Server has the Transact-SQL programming language for stored procedures and functions that we learned about in 4th semester. From what I can read, MySQL also supports stored procedures , functions and triggers, but this is something new in the latest version, and there is very little documentation available on how it works on the Internet. Here MSSQL Server has a clear advantage, but for my project I am not using stored procedures (and my experience with them from 4th semester is also that stored procedures are not so easy to debug and maintain, so better to work with normal queries to the database).

Working with MySQL

Working with MySQL seems much faster to work with on my PC, maybe because it has less system requirements. It has the necessary tools, so it is easy to work with the database, and the SQL syntax has minimal difference, so if you know SQL for MSSQL Server, then you can apply the same knowledge easily on MySQL. I think in my specific project, there would be no real difference if I work with MySQL or MSSQL Server.

Iteration Planning

Iteration planning is much like the traditional analysis part in XP. First the customer and I select which user stories that are going to be implemented for the coming iteration, based on the calculations on project velocity from the previous iteration. For the first iteration, we just used my estimates as truth, then later we can modify depending on how the first iteration went. This is very smart with XP I think.

For each of the selected user stories, it must be analyzed, and from the user stories, tasks are created. One task is equivalent of a more detailed description of one part of the solution to the user story. These tasks contains more knowledge now when we have analyzed the problem, so they also contain a more precise estimate for the tasks.

Below I have listed the tasks that were created during the project for each iteration, together with estimates and follow-up for their real time spent, this allows me to do the project velocity measurement.

INo	TNo	UNo	Task name and Description	Time estimate	Actual time
1	1	1	**Setup Django Project**: A Django project and Django application has to be created.	5 hrs	3 hours
1	2	1	**Create a stock property model** Create stock properties with fields such as Name,URL, XML path, convert expression and also has to create the table in database.	3 hrs	4 hours
1	3	2	**Create the stock property for three fields** Create the stock property with for the fields such as PE/Ratio, Shares out, Div_Yield.	2 hrs	1 hour
1	4	2	**Create a model to hold values** A model has to be created, which can hold information on the values extracted, and the symbols which they belong to.	3 hrs	1 day
1	5	2	**Extract values and Save in database** Save the values into database after download the URL, and execute XML path and converter expression.	1 day	2 days
1	9	2	**Create command to extract values for one symbol**	1 hour	5 hrs

			There should be an admin command that given a symbol extracts all fields for that symbol.		
1	6	3	**Calculate X,Y coordinates for Distribution graph** Group the companies by values from database. So values can fit to the graph.	1 day	1 day
1	7	3	**Create image by the graph data** The calculated graph data has to be plotted in an image, the image has to show the data, with X axis legend.	5 hrs	1 day
1	8	3	**Create the URL to show the graph** Create a view that takes a stock property ID and returns the graph image for it and then create a URL mapping for a view.	2 hrs	3 hours

INo	TNo	UNo	Task name and Description	Time estimate	Actual time
2	10	4	**Get a list of symbols** Freyr has a web service,where list of all symbols available. I have to get the code from that web service and return it as list.	1 day	1 day
2	11	4	**Create admin command to perform import** Create an admin command that can be scheduled by cron to run periodically. The admin command should iterate over the symbols and extract all properties for each symbol	2 days	1 day
2	12	6	**Create Model for StockSymbol** StockSymbol has to hold the values from the Freyr webservice in a db. So the program doesn't need to contact Freyr server for the values every request.	3 hrs	2 hours
2	13	6	**Create an admin command** Create an admin command that can download the list of symbols and save into database.	2 hrs	1 day
2	14	6	**Create a view, form and template for the SearchForm** Form has to contain fields discussed. View has to return the response including the form and template. Template has to show the form correctly as in the prototype.	1day	2 days
2	15	6	**View to generate input fields for one stock property** View has to create a html table row with input fields for the property by given stock property id.	2 hrs	1 hour
2	16	6	**Re-factor the stock property value** SymbolsClient should remove the old method for	1 day	1day

			get_symbols every where it is used should be updated ImportAll admin command has to be changed to all symbols client and then load from the database. Database has to be recreated.		
3	17	7	**Create query for performing search** A SQL query has to be created that can find companies that match a set of search criteria. The query needs to be exposed in a class so it can be used by other objects.	2 hours	3.hrs
3	18	7	**Create view, template and form for result** A search form has to be created which can receive the input submitted from the user. A view has to be created to take the input from the form and pass its details to the search functionality. Finally, a template has to be created which can show the search results in a table, with a dynamic set of stock properties.	5 hours	4.hrs
3	19	7	**Re-factor search form to use new result view** In the GUI, the search form template has to be modified so it will send data to the show-result view.	1 day	3.hrs
3	20	8	**Create the model for history** Historical values has to be saved in database so model has to be created.	2 days	2 days
3	21	8	**Re-factor the extractor to save the history** The extractor module should save historical values when a new value is found while extracting.	1 day	1 day

INo - iteration number, TNo - task number, and UNo - user story number.

Note:- In **Appendix-G**, the working papers for the tasks have been attached.

Design

In the design, the goal is to take the requirements and the analysis, and to make a design for the software, where I try to plan how the implementation should be. It is about deciding the architecture of the program on the high level, and to design about classes and interactions in the low level.

For my project I have used XP, so the design is also an iterative process. What I will present here is the final design, the outcome of the 3rd iteration.

Note:- Please see the **Appendix-D** for intermediate design documents.

Selecting architecture

For my project I have chosen to work with the Django framework. Django already has an architecture, which they call Model-View-Template (MVT). How I see MVT, it is the same as Model-View-Controller (MVC), only with different names.

Model in Django and in MVC is almost the same. In Django models are too simple, they only work with the data, not so much with operations, but, I think that it is possible to make it work like that.

The view in Django works almost the same as the controller in MVC. It handles the input from the users, and decides what actions to take. From my understanding this is exactly what the controller in MVC does.

The template in Django works much like the view in MVC. It handles the GUI parts. In a web-application that is creating the HTML for the GUI.

There is a difference, in Django it is not possible to work with observing. So, the view does not observe the model, and the controller does not observe the view. It happens by HTTP interactions, and views are created/destroyed all the time.
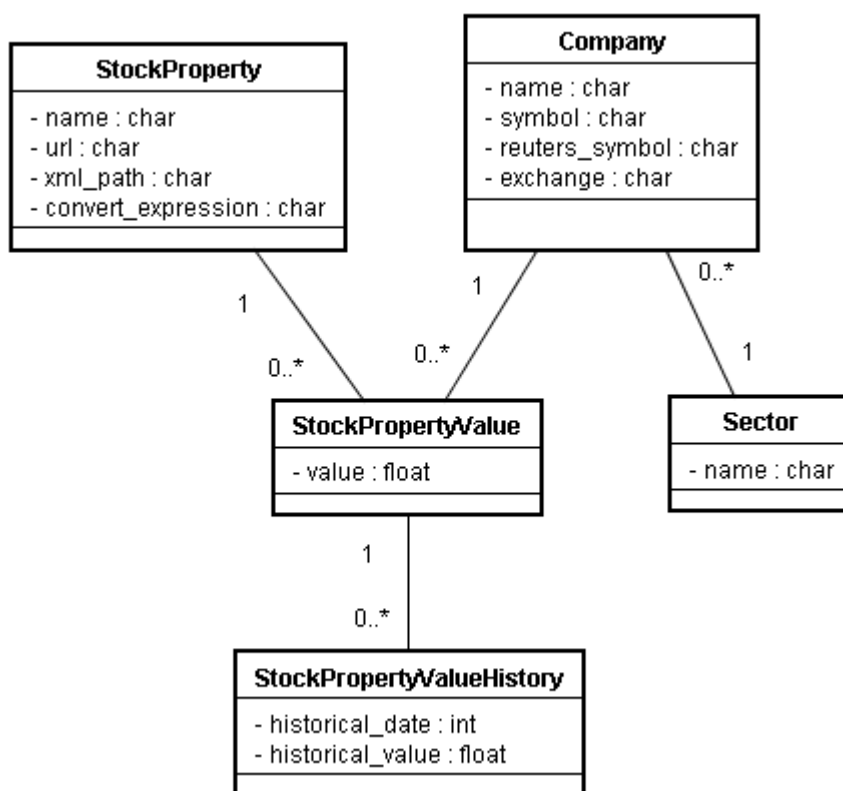
So, my project is working with something like the 3-layer MVC architecture.

UML modeling

XP does not require me to use UML, but since it is the tool that I know most for modeling, then I have created all the models in the UML language.

Business object model

First we have the business object model, which shows the main classes of the system, and how they are related together.

We see that a company can have relations to some values, that are related to some property names. For example the stock property "Price/Earnings Ratio" could be related to the company "IBM" by the value of "10.0". Also, a company belongs to a Sector, which could be for example "IT Services" in the case of IBM. Also we have the historical values, which are related to a current value of *StockPropertyValue.*

This model is modeled little different than what I first imagined. I could imagine that it would be a good idea to have one table called StockPropertyValues, which had many attributes (instead of just one), for example pe_ratio, shares_out, market_cap, etc .. But, it is modeled like this because of the requirement to add new attributes on the fly, which is not possible the other way.

Design object model

In the business object model, we have only the main classes of the system. In the design object model, I have a more detailed design, where I look at exactly how am I going to implement and organize my code.
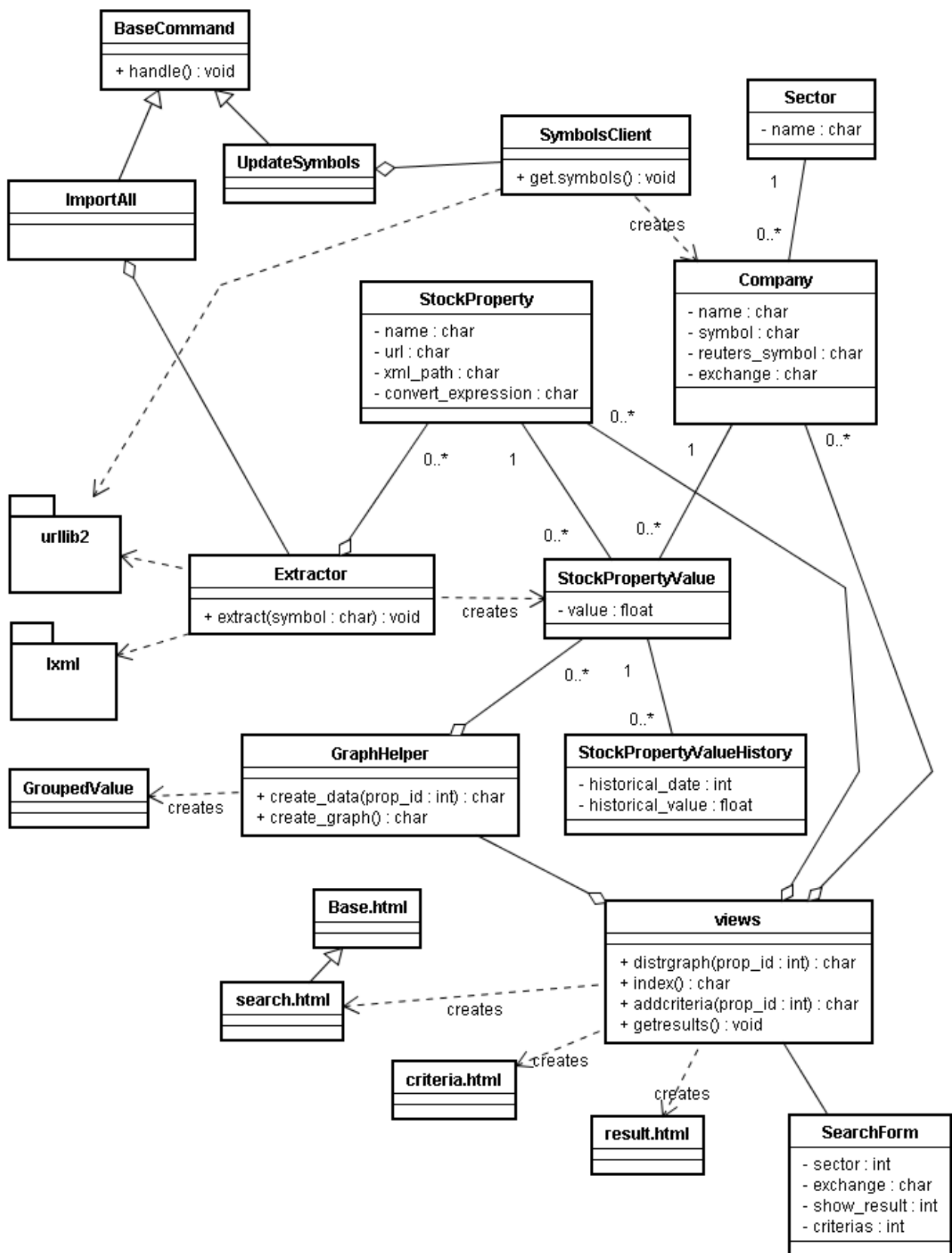
While it looks complex, it is actually quite simple, if I try to make a simplified version it looks like this:
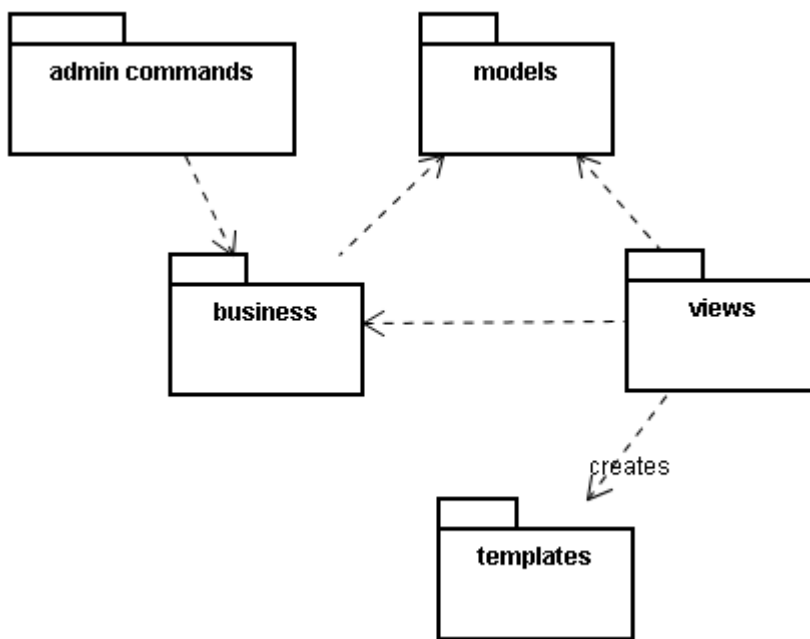
There are some main areas to be identified in this DOM, depending on how we view it.

- ImportAll and UpdateSymbols classes are both classes that provide the integration of my code with the administration tool provided by the Django framework.

- The same with views, it is providing the integration between my code and the web GUI which is using Django's MVC model.

- Extractor is one of the main classes in my project, it handles the extraction of values from Freyr database, it has dependencies on packages urllib2 and lxml which are provided by Python, and allows me to access HTTP pages as files and parse and work with XML documents.

- SymbolsClient handles the interaction with the Freyr web service.

Database modeling

When I am working with Django, database modeling happens in a different way than usual. How I would normally work is by the following steps:

1) Create E/R diagram.

2) Create E/R → relational mapping

3) Create relational mapping → SQL schema.

However, in this project I am working with an object relational mapper (ORM), which handles the SQL schemas for me, so my procedure is now:

1) Create and implement the UML model

2) Run the SQL schema generator.

3) Use a reverse-engineering tool to create E/R from an existing schema.

Creating models

First I implement the model in Python by extending the Django Models API, as shown in the example below. I must specify the data-types for each of the attributes.

StockProperty and StockPropertyValue models from models.py

```python
class StockProperty(models.Model):

    name = models.CharField(max_length=100)
    url = models.URLField()
    xml_path = models.CharField(max_length=250)
    convert_expression = models.CharField(max_length=250)

    def __str__(self):
        return self.name

class StockPropertyValue(models.Model):

    symbol = models.CharField(max_length=20)
    value = models.DecimalField(max_digits=19, decimal_places=5)
    stock_property = models.ForeignKey(StockProperty)

    def __str__(self):
        return "%s(%s)=%.2f" % (str(self.stock_property), str(self.symbol), self.value)
```
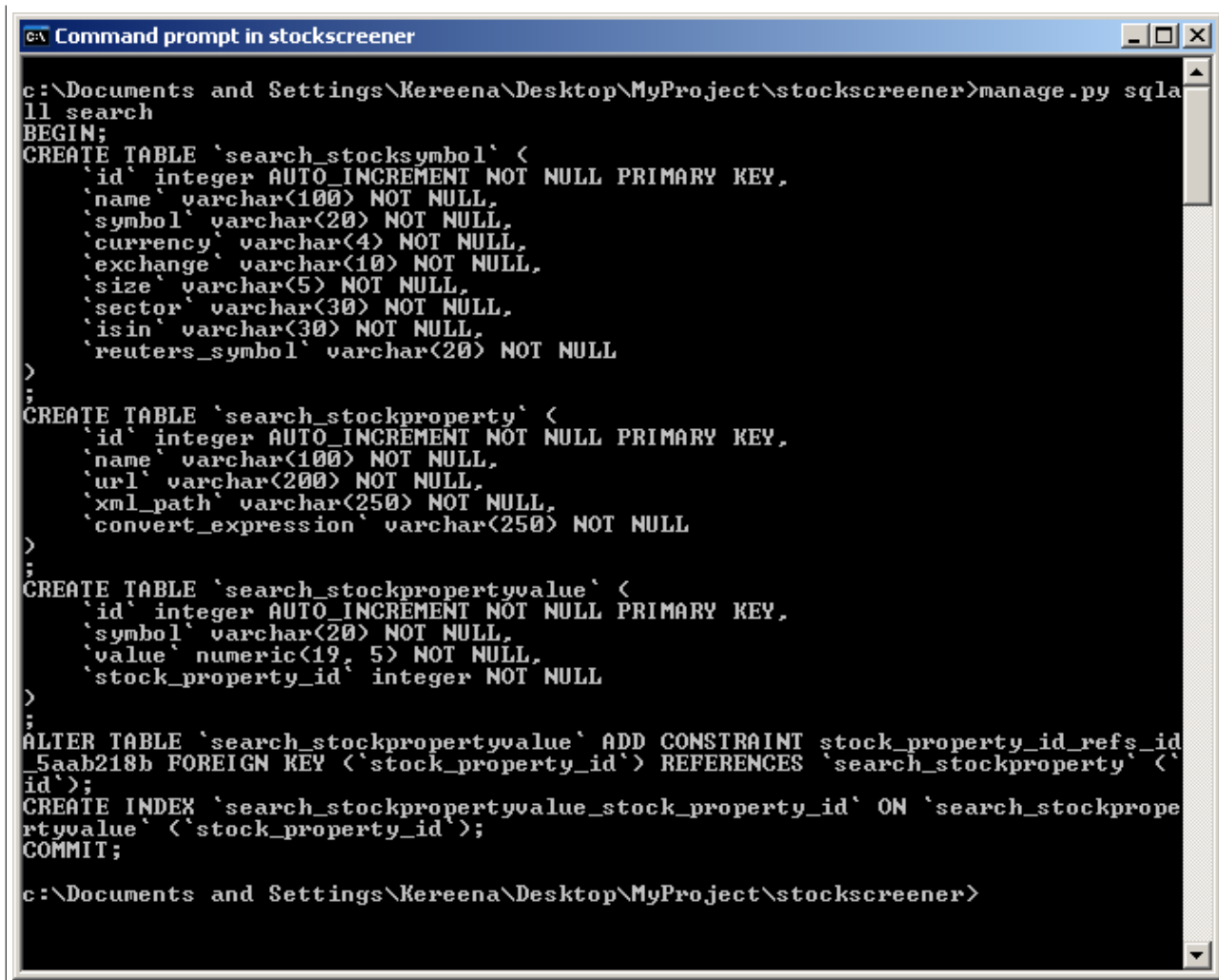
When I have created the models, I invoke the Django management command to generate the SQL schema. This schema I will then load into the database by any SQL tool (ie. MySQL Query Browser).

Running manage.py to generate schema

```
Command prompt in stockscreener                                    _ □ X

c:\Documents and Settings\Kereena\Desktop\MyProject\stockscreener>manage.py sqla
ll search
BEGIN;
CREATE TABLE `search_stocksymbol` (
    `id` integer AUTO_INCREMENT NOT NULL PRIMARY KEY,
    `name` varchar(100) NOT NULL,
    `symbol` varchar(20) NOT NULL,
    `currency` varchar(4) NOT NULL,
    `exchange` varchar(10) NOT NULL,
    `size` varchar(5) NOT NULL,
    `sector` varchar(30) NOT NULL,
    `isin` varchar(30) NOT NULL,
    `reuters_symbol` varchar(20) NOT NULL
)
;
CREATE TABLE `search_stockproperty` (
    `id` integer AUTO_INCREMENT NOT NULL PRIMARY KEY,
    `name` varchar(100) NOT NULL,
    `url` varchar(200) NOT NULL,
    `xml_path` varchar(250) NOT NULL,
    `convert_expression` varchar(250) NOT NULL
)
;
CREATE TABLE `search_stockpropertyvalue` (
    `id` integer AUTO_INCREMENT NOT NULL PRIMARY KEY,
    `symbol` varchar(20) NOT NULL,
    `value` numeric(19, 5) NOT NULL,
    `stock_property_id` integer NOT NULL
)
;
ALTER TABLE `search_stockpropertyvalue` ADD CONSTRAINT stock_property_id_refs_id
_5aab218b FOREIGN KEY (`stock_property_id`) REFERENCES `search_stockproperty` (`
id`);
CREATE INDEX `search_stockpropertyvalue_stock_property_id` ON `search_stockprope
rtyvalue` (`stock_property_id`);
COMMIT;

c:\Documents and Settings\Kereena\Desktop\MyProject\stockscreener>
```
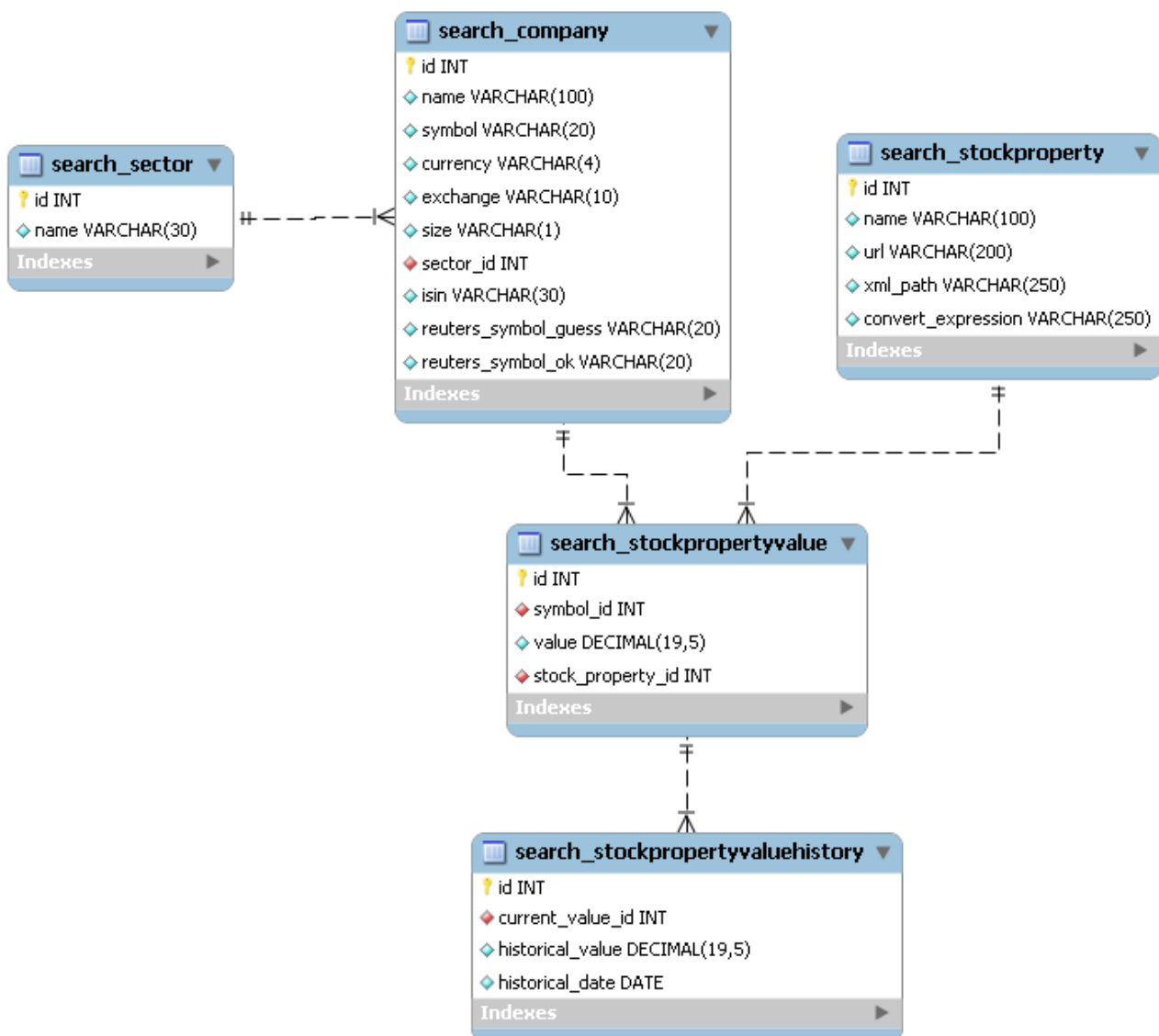
This example shows also how the table names are created, eks for my class StockProperty which belongs to the application "search", the generated table name is "search_stockproperty".

Finally, I can run the reverse-engineering tool to generate the E/R diagram, as shown below.

I think that it is still useful to have the E/R diagram, since it gives a better visual presentation of how the database is laid out than just looking at tables and code in models.

Using this approach, the database gets modeled differently that how I would maybe normally model it, for example: All tables get a primary key "id" as INT. Eg. The stock property value, I would maybe have selected differently. But I think it is good enough for using.

The tables are equal to my UML model, so if my UML model is good then the resulting database should also be fine. Only, I saved the steps of manually performing E/R diagrams.

Implementation

For implementation, I understand the part where I use my design to create the actual code for the working system. Since I have chosen XP process model, I will be working in iterations, therefore I will describe little about what has happened for implementation in each iteration. In XP testing is also part of the implementation, but in my report I have kept these apart, so testing is shown in the Testing chapter.

Iteration 1

For Iteration 1 the main focus was the following two:

- Get data into database by extracting system.

- Create a distribution graph from the data to show on the web-page.

Extracting data from Reuters

The process of getting data into the database involved reading web-pages from Reuters, extracting values from the web-pages, and saving the values in the database, in a way which is meaningful to the system. Eg the string "10M" on a web-page should be saved as the number 10000000 in the database. For this purpose I have created an Extractor class, which handles the steps involved:

1. Download the web-page

2. Find the value that is interesting in the web-page.

3. Convert the value to something meaningful.

4. Save the converted value into the database.

Below here, we will see the code for each of the steps (the full extractor.py code can be found in **Appendix -A**).

Extracts from extractor.py, download method

```
# download the url and save it in object
def download(self, url):
    opener = get_opener()
    fd = opener.open(url)
    html = fd.read()
    fd.close()
    return html
```

The page is downloaded using an "opener", which is a Python implementation of a client to download arbitrary URLs. So we see that this type of operation is very simple with Python.

Extracts from extractor.py, extract method

```
# evaluate the xpath and save it in object.
def execute_xpath(self, html, xml_path):
    # parse the html
    doc = ElementSoup.parse(StringIO.StringIO(html))
    elem = doc.xpath(xml_path)

    # if there is no element, return None, otherwise return the contents.
    if elem is None or len(elem) == 0:
        return None
    else:
        return elem[0].text
```

The value is found from the downloaded page by evaluating an XPath onto the page. First I have to create an XML structure from the HTML document, which we can use. For this purpose I am using the ElementSoup library. ElementSoup is a mixing of two libraries:

- BeautifulSoup: A very flexible parser of HTML. It can handle errors in HTML and still produce a parse-tree

- ElementTree: One of the XML structure libraries in Python (I use "minidom" later, which is

another, simpler, XML library). In my UML this shows as lxml, because the implementation is called that.

When I have the XML structure, it is easy to call the xpath method on it, to have it return a value, if found in the document. An example of the XPath is: (.//td[@class='data'])[4] which means:

```
take the 4th TD element that has the attribute "class" set to the value "data"
```

For example this document it will pick the value 123:

Example XML structure
```
<html>
<body>
  <table>
    <tr>
      <td class="data">12.5</td>
      <td class="data">1000</td>
      <td class="data">444</td>
      <td class="data">123</td>
    </tr>
  </table>
</body>
</html>
```

Working with XML in Python is simple, but there are many libraries, and it is little hard to decide which one to use for an exact purpose. With help from Freyr, I found this ElementSoup, which is fit good for my purpose.

The last step is to convert the extracted string into a value (decimal number) that can be saved into the database.

Extracts from extractor.py – the execute_converter method
```
    def execute_converter(self, value,stock_property):
        # setup ssi function in local scope
        SSI = convert_ssi_units
        # setup USD function in local scop
        USD = usd_to_value
        # setup value as 'x' variable
        x = value
        # evaluate x using the convert_expession of the the stock property
        converted = eval(stock_property.convert_expression)
        # return as decimal, if it is already, then do nothing, otherwise try to convert
        try:
            if isinstance(converted, decimal.Decimal):
                return converted
            else:
                return decimal.Decimal(str(converted).replace(',',''))
        except:
            return None
```

Here we see that I use one of the advanced functions in Python, which is to evaluate a string. That means, Python will compile the string, and run it, from within my program. This is a feature I think C# does not offer, but it is also little strange to use, because there is no compile time check of my code in the string. We also see that I use some helper functions SSI and USD. Because everything is objects in Python, also methods, I can just assign them to variables and use them.

Extracts from extractor.py – the convert_ssi_units method
```
def convert_ssi_units(value):
    """
```

```
    convert SSI units handles converting from SSI units (k=kilo, m=mega, g=giga) to
    a number I can save in the database.

    Eg. it expands something like: 1234k => 1234000 or 1M => 1000000
    """
    ssi = {'k': decimal.Decimal(1000),
           'm': decimal.Decimal(1000000),
           'g': decimal.Decimal(1000000000)}

    # use Regular expresssion.
    # [0-9.,]+ matches numbers and . and , - eg. 19,123.20 or 19 or 123,220 or 12.000
    # [MmGgKk] matches 'M' or 'm' or 'G' or 'g' or 'K' or 'k'
    # " *" matches 0 or more spaces.
    # ( ) means to extract the values, it is extracting a number and a unit
    m = re.match('^([0-9.,]+) *([MmGgKk])', value)

    if m is None:
        return value
    else:
        # get the extracted values
        val, unit = m.group(1), m.group(2)
        return decimal.Decimal(val.replace(',','')) * ssi[unit.lower()]
```

As seen the function converts between SSI units into a number.

Creating distribution graph

For the second part of this iteration, I implemented functionality to show a graph of the distribution of companies given a stock property. One example could be: How is the distribution of the Shares Out for all companies.

For this task I used GoogleChart API to draw the graph. It is simple to use in a web-application, and I do not spend time on the complex part of drawing graphical images. My task is then reduced into producing the data required for the GoogleChart API to work, and generating the call to the API.

Extracts from DistrGraph.py – create_data method.

```
        # find minimum and maximum values in the database
        values =
StockPropertyValue.objects.filter(stock_property__id=stock_property_id).values_list('val
ue',flat=True)

        a = min(values)
        b = max(values)

        # calculate how big is each x
        diff = (b - a) / x_size
```

Here we see how I use the StockPropertyValue model to find a list of all the values for a specific Stock Property, and then using Python's min/max methods to get min and max values. I decided here only to use Django's models for this feature.

Django and Python write that it will handle things "smart", but I think to write a custom SQL query using **MAX(value)** and **GROUP BY stock_property_id** will be faster. After I did this part of the code, I found how that can be done easily, but it is hard to make it work on both development database SQLite3 and MySQL, the SQL is little different for each.

Extracts from DistrGraph.py -

```
        # find grouped values for each range
        for i in range(x_size):
            # calculate end range for this grouped value
```

```
            current_end_range = current_start_range + diff

            # ask database count of companies in database with value inside this range
            number_of_companies = StockPropertyValue.objects.filter(stock_property__id =
stock_property_id,
                                                            value__gte =
current_start_range,
                                                            value__lte =
current_end_range).count()
```

Here we see one of the part where I calculate how many companies are inside a given range. For this Django has a *count()* method on a query, which can optimize the SQL, so this is very fast. *value__gte* is a way of saying that value must be greater than or equal to, likewise with *value__lte* means less than or equal to.

For calling the GoogleChart API, the only thing to do is to build an URL which express the data to show. I have not included that here, but it can be found in DistrGraph.py in **Appendix-A**.

Iteration 2

For second iteration, my focus was on producing the user GUI parts and how the user should interact with the system. I created a prototype, which can be found on the CD in the "prototype" folder.

The main points for creating the user parts this one was:

·   Create prototype GUI without functionality.

·   Implement the search form of the GUI using Django.

·   Adding and removing search criteria in the GUI.

·   Working with Freyr's web-service.

Designing a prototype GUI

By the advice of Freyr, I built the GUI using simple HTML, and then applying CSS to make it look good. As above I started with a simple prototype, without functionality. This way it was possible for Freyr to see how I imagine the GUI to work, and they could come with useful input on how I could make it.

HTML output from Search form (cleaned up to be less space)
```
<div class="title">
  Search
</div>
<form method="post" action="/search/result/" onsubmit="return search('results', this);
">
<div class="rightdrop">
  <select name="sector" id="id_sector">
    <option value="" selected="selected">Select sector ...</option>
    <option value="1">Consumer Discretionary</option>
  </select>
  <select name="exchange" id="id_exchange">
    <option value="" selected="selected">Select exchange ...</option>
    <option value="CSE">Copenhagen Stock Exchange</option>
  </select>
</div>
```

```
<div class="criteria">
  Please select which criteria you want to screen the stocks for ....
  <table width="100%" id='id_criterias'>
    <tr
class="head"><th> </th><th>Criteria</th><th>Min</th><th>Max</th><th>Distribution</t
h></tr>


  </table>
  <br />
  <select onchange="addCriteria(&#39;id_criterias&#39;, this);" name="add_criteria"
id="id_add_criteria">
    <option value="" selected="selected">Select to add criteria ...</option>
    <option value="1">Price/Earnings Ratio</option>
  </select>
</div>
<div class="rightdrop">
  <select name="show_result" id="id_show_result">
    <option value="criteria">Show only selected in result</option>
    <option value="all">Show all in result</option>
  </select>
</div>
<div style="float:left;">
  <input type="submit" value="Search now" />
</div>
</form>
```

As we can see, there is very little layout information (information on how it should look like) in the HTML code. What I am using are *div* (divider) tags, which are like building blocks, and assign them a *class*. By using CSS I can then put style on how each class should look like, and how it should be positioned in the page as seen in the example below:

Extract from style.css – some different styles

```
tr.head th {
        border-bottom: 1px #000 solid;
}
.criteria {
        padding-right: 300px;
}
.rightdrop {
        float: right;
        margin: 10px;
}
.search {
        margin: 10px;
}


.title {
        background: #ccf;
        padding-left: 10px;
        font-weight: bold;
        border-bottom: 2px #99f solid;
}
```

As the example I have created two screen-shots of how the form looks without and with the CSS style applied:

Without CSS	With CSS

As we can see, there is a big difference in the looks, but the page is still usable without. I think it is a good idea to separate layout and style like this, it is easy if I need to modify a color, I can just do it in the CSS without editing the HTML templates.

Implementing search form using Django

In order to implement the prototype in Django, there are some steps that must be completed, as discussed in the chapter comparing ASP.NET with Django.

- Create a view that decides how to handle the request.

- Create a form that can handle the input from user.

- Create a template that contains the information to show to user.

For the view there are actually also two parts, one is to create the view and another is to create the mapping to the view. We recall that in Django, "view" is just another word for "controller" in the MVC architecture.

To create a view is just to create a method inside a file called *views.py*. The method can actually be anywhere, but this is just to follow conventions.

Extract from views.py – method to create search page.

```
def index(request):
    form = forms.SearchForm()

    return render_to_response('search/search.html', {'form': form})
```

We can see that the only thing the view does, is to create the form object, which can handle input from user, and then to use the Django method *render_to_response*, which loads the specified template *search/search.html* with the arguments given, the form.

We can see in the file urls.py how the mapping is created to the method *index* in the *stockscreener.search.views* module. It has been highlighted with bold what is related to the search form view.

Extract from urls.py – showing how URLs are mapped to methods in views module.

```
urlpatterns = patterns('',
                    (r'^admin/(.*)', admin.site.root),
                    (r'^distrgraph/(?P<stock_property_id>\d+)/$',
'stockscreener.search.views.distrgraph'),
                    (r'^search/$', 'stockscreener.search.views.index'),
                    (r'^search/criteria/(?P<stock_property_id>\d+)/$',
```

```
'stockscreener.search.views.addcriteria'),
                    (r'^search/result/$', 'stockscreener.search.views.getresult'),
)
```

Next we need to create the form, which is used by the view, to handle the input of the user. Using the form I can define which are the attributes to enter, and also I can fill out the form with some default data, in my case I want to show for example sectors in the form, presented by a drop down menu in the GUI.

Extract from forms.py – SearchForm class

```python
class SearchForm(forms.Form):
    SHOW_RESULT = (
        ('criteria', 'Show only selected in result'),
        ('all', 'Show all in result'))

    sector = forms.ChoiceField(choices=load_sectors(), required=False, initial='')
    exchange = forms.ChoiceField(choices=load_exchanges(), required=False, initial='')

    add_criteria = forms.ChoiceField(choices=load_criterias(), required=False,
initial='')
    add_criteria.widget.attrs['onchange'] = "addCriteria('id_criterias', this);"

    show_result = forms.ChoiceField(choices=SHOW_RESULT, required=True)

    minmax_criteria = None
```

We can see that for each field I define what type the field is. In this case all are choice fields, meaning the user has to give one of the available choices. I also define that none of the attributes are required, so user does not need to enter anything. For the add_criteria part, I need some special handling, which we will see about why little later. We can also see some "load" functions, which are providing the choices for the user. One example load function can be seen here:

Extract from forms.py – load_sectors

```python
def load_sectors():
    """
    Helper function to load a list of sectors.
    """
    # load sectors and convert to a list: ((1,'Sector1'), (2,'Sector2'))
    l = [ (s.id, s.name) for s in Sector.objects.all() ]
    # add empty sector to the beginning of list (starting in dropdown menu)
    l.insert(0, ('', 'Select sector ...'))
    return l
```

Finally, *minmax_criteria* I do not handle by Djangos form framework, it requires special handling. The problem is that it does not handle specifying a list of properties with dynamic length.

To solve this, I created the criteria in the form as *min[stock_property_id]* and *max[stock_property_id]*, and then a method that can handle to parse these attributes into a dynamic list of criteria. We can see here below how the values are extracted from the HTTP request.

Extract from forms.py – find_minmax_criteria method

```python
    def find_minmax_criteria(self, data):
        """
        Find all minmax criterias from the search form.
        """
        found = {}
        data = dict(data)
        for k in data.keys():
```

```
                m = re.match(r'(?P<minmax>min|max)\[(?P<property_id>\d+)\]', k)
                if m is not None:
                    minmax = m.group('minmax')
                    property_id = int(m.group('property_id'))
                    if not found.has_key(property_id):
                        found[property_id] = MinMaxCriteria(property_id)
                    if minmax == 'min':
                        found[property_id].min_value = self.__to_value(data[k])
                    elif minmax == 'max':
                        found[property_id].max_value = self.__to_value(data[k])
        self.minmax_criteria = found
```

What is done is to loop over all the values submitted by the user, and use a regular expression to find those that are matching: *(min|max)[number]* (highlighted by bold in code). Then create the *MinMaxCriteria* objects using this information.

We have already seen the output of the template from our design of the GUI. Now the next step is to create a Django template which will show the GUI in the same way. As we saw for the view, we send one argument to the template, which is the form.

Template search/search.html – using form components in template.

```
{% extends "base.html" %}
{% block title %}Search{% endblock %}
{% block extrahead %}{{ form.media }}{% endblock %}
{% block content %}
<form method="post" action="/search/result/" onsubmit="return search('results', this);
">
  <div class="rightdrop">
    {{ form.sector }}
    {{ form.exchange }}
  </div>
  <div class="criteria">
    Please select which criteria you want to screen the stocks for.
    You can select as many criteria as you want.
    <table width="100%" id='id_criterias'>
      <tr class="head">
      <th> </th>
      <th>Criteria</th>
      <th>Min</th>
      <th>Max</th>
      <th>Distribution</th>
      </tr>
    </table>
    <br />
    {{ form.add_criteria }}
  </div>
  <div class="rightdrop">
    {{ form.show_result }}
  </div>
  <div style="float:left;">
    <input type="submit" value="Search now" />
  </div>
</form>
<div style="clear:both;" />
<div class="title">
  Results
</div>
<div class="result" id="results">
</div>
{% endblock %}
```

We can see that this is very simple. Using the form variable, we can ask its attributes, and they will represent themselves as HTML form elements.

Using AJAX to update GUI

In web applications it is normally so that I have to send one HTML page to the user, the browser will render this page, and the user can see the GUI. But, in my project, I would like to change the GUI without changing the whole page. I want to do this because my application is stateless, that means on the server, I do not keep information about what the user is doing presently. I would like the state to be only in the GUI, that means in the browser of the user.

In order to make my application like this, I want to be able to add and remove the criteria from the GUI, without creating the full HTML page again.

For this purpose, I am using AJAX, which means Asynchronous Javascript And XML. I have created a small sequence diagram that shows how I imagine it will work:
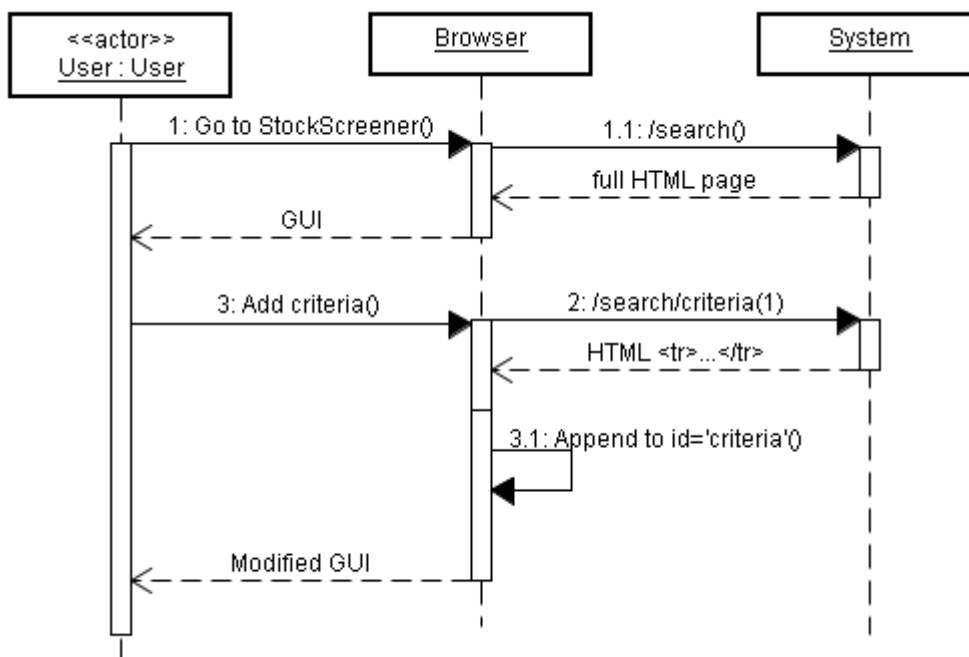
If we recall the template above, there is one table with the *id='id_criterias'*. It is this criteria that I want to update. Also if we recall the SearchForm class, we see that I add an onchange attribute with a javascript function to the *add_criteria* attribute. On change, means that the function will be executed when the user changes the value of the drop-down box, meaning the user selects some criteria to add.

Extract from site_media/stockscreener.js – addCriteria function.

```
function addCriteria(tableId, selected) {
        // select table, go down to the TR level, and insert the new row After.
        new Ajax.Request('/search/criteria/'+selected.value+'/', {
                method: 'GET',
                onSuccess: function(transport) {
                        $(tableId).down('tr').insert({after: transport.responseText});
                },
                onException: function(transport, e) {
                        alert('Error! ' + e);
                }
        });
}
```

What this method is doing, is that it is creating a new AJAX request, and on success, I specify that it

updates the table, by adding a the content of the response after the *TR* tag inside the table. If there is an exception, it will show an alert pop-up box with the error message.

The *Ajax.Request* object is coming from a library called prototype.js. It simplifies much of the Javascript, by providing functionality that works in the same way, no matter which browser the user is using. The library can be found in the site_media folder. See **Appendix-F** for more information.

Working with web-service

The web-service that Freyr provides is a simple web-service. One URL gives some XML back, that I need to parse. Here I needed to create a client for this XML. I could not use the standard web-service tools, since it is not a "real" web-service (I am not sure what a real web-service exactly is, but for example, this web-service has no WSDL, that I remember is required from C#).

Extract from extractor.py – parts of SymbolsClient class

```
        document = xml.dom.minidom.parse(self.get_document_fd())

        # for each "company" element in the XML
        for element in document.getElementsByTagName("company"):

            # get values from the "company" XML
            name = self.__elem_value(element, 'name')
            currency = self.__elem_value(element, 'currency')
            exchange = self.__elem_value(element, 'exchange')
            size = self.__elem_value(element, 'size')
            sector = self.__elem_value(element, 'sector')
            symbol = self.__elem_value(element, 'symbol')
            isin = self.__elem_value(element, 'isin')
            reuters_symbol_guess = self.__elem_value(element, 'reuters-symbol-guess')

            # if some value is not there, then continue to next "company"
            if name is None or currency is None or exchange is None or size is None \
                    or sector is None or symbol is None or isin is None \
                    or reuters_symbol_guess is None:
                continue
```

From the XML, I can ask to get all *company* tags, and for each of them I can extract values from the tag. This works good since the Freyr XML structure looks like this:

Freyr XML web-service structure

```
<companies>
  <company>
    <name>IBM</name>
    <currency>DKK</currency>
    <exchange>CSE</exchange>
    ...
  </company>
  <company>
    <name>MAERSK</name>
    <currency>DKK</currency>
    <exchange>CSE</exchange>
    ...
  </company>
</companies>
```

When I have the XML converted into Company and Sector model objects, it is easy to save them to the database.

Iteration 3

In Iteration 3, my focus was on making the final parts of the system, that is to get the result to show. Also for this iteration, my customer decided that if any time left, the part with adding historical data to the database model. I found time to do both. The main points for this iteration are:

- Constructing query to find search results

- Adding functionality in the GUI to show results

- Adding the historical data.

Querying the database

Because I am using XP model, I have not thought about how I must query the database, therefore, my database model is maybe not so good fit for the querying. When I did this approach, I found that it is not good fit, by the practical way, that the query was not so easy to write.

If I recall my database model, the problem was, that the values in my database are not found in one row in the database, but are found in more rows, and it is the sum of these rows that is interesting.

When I realized this, it was easier to create the query, and also because MySQL database supports nested queries. If it did not support this, I would have to either change the database model, or, handle some parts of the querying in Python, which means probably slower queries for the user.

Extract from search.py – the SQL query

```
SEARCH_QUERY = """
      select t1.id from (
          select c.id as id, count(*) as number
            from search_company c left outer join
                search_stockpropertyvalue v on c.id = v.symbol_id
                left outer join search_stockproperty p on p.id = v.stock_property_id
          where ( CRITERIA )
              SECTOR
              EXCHANGE
            group by c.id
      ) t1
      where number >= CRITERIA_COUNT
      order by number
      limit 25
"""
```

There is the nested query, which selects the id of the company, and a count of how many criterias are matching for this company. Then the outside query is selecting out of those, the companies that is matching all the criterias by checking *number >= CRITERIA_COUNT*. As we can see, there are some bold elements that needs to be replaced while querying.

Extract from search.py – parts of query method

```
def query(criterias, sector=None, exchange=None, show='all'):

    # get database connection
    from django.db import connection
    cursor = connection.cursor()
    qn = connection.ops.quote_name

    # build criteria
    if len(criterias) == 0:
        sql_criteria = '1'
    else:
```

```
        sql_criteria = ' or '.join([ c.to_sql(qn) for c in criterias ])

    sql = SEARCH_QUERY.replace('CRITERIA_COUNT',
str(len(criterias))).replace('CRITERIA', sql_criteria)
    sql = sql.replace('SECTOR', build_sector_criteria(qn, sector))
    sql = sql.replace('EXCHANGE', build_exchange_criteria(qn, exchange))

    cursor.execute(sql)
```

The query method handles this. It is replacing each of the "variables" in the query with some SQL created dynamically. As we recall from earlier, the *criterias* in the method is a list of *MinMaxCriteria*, which can provide SQL for the criteria.

Extract from search.py – to_sql method from MinMaxCriteria class

```
    def to_sql(self, qn_func):
        if self.min_value is None and self.max_value is None:
            return "( p.id = %d )" % (self.stock_property_id)
        else:
            return "( p.id = %d%s%s )" % \
                (self.stock_property_id,
                 self.__has_value(self.min_value, '>='),
                 self.__has_value(self.max_value, '<='))

    def __has_value(self, value, operator):
        if value is None:
            return ""
        else:
            return " and v.value %s %s" % (operator, str(value))
```

The bold parts are where the criteria is created. If no min or max value has been specified, it means we want that criteria, but we don't care about the values. If min or max has been specified, then we use the *__has_value* method which creates a *<= or >=* criteria for the value.

The sector and exchange criterias are handled very easily also with helper functions.

Extract from search.py – build_sector_criteria function

```
def build_sector_criteria(qn, sector):
    if sector is None or sector.strip() == '':
        return ""
    else:
        return " and c.sector_id = %s " % sector
```

If there is no value, ignore the criteria, and if there is a value, then build criteria that matches a specific sector.

When the program is running, we can see an example of the constructed SQL in console. For example here I query on two criterias (P/E Ratio and Shares out), and the sector "Telecommunications" (id=9) and which are traded on the STO (Stockholm) exchange.

Example real query:

```
    select t1.id from (
        select c.id, p.name, v.value, count(*) as number
          from search_company c left outer join
                search_stockpropertyvalue v on c.id = v.symbol_id
                left outer join search_stockproperty p on p.id = v.stock_property_id
          where ( ( p.id = 1 and v.value >= 0.44 and v.value <= 20 )
                 or ( p.id = 2 and v.value >= 30000 and v.value <= 200000000 ) )
            and c.sector_id = 9
            and c.exchange = 'STO'
          group by c.id
    ) t1
```

```
where number >= 2
order by number
limit 25
```

We can notice the *limit 25* which means to return only 25 results at a time. It could be a good idea to implement paging here, so we could return more. Also we notice the *order by number* which means that we return those that match the most criteria first.

This first query only returns the database ids of the companies that the user is interested in. The second part of the querying is to load those companies from the database, including the criteria that the user wants to have displayed.

Extract from search.py – part of query method

```
    # get the headers to show
    if show == 'all':
        # show all properties
        headers = StockProperty.objects.all()
    else:
        # show criterias as properties
        headers = [ StockProperty.objects.get(id=c.stock_property_id) for c in criterias
]

    result_list = []
    # load data including headers
    for row in cursor.fetchall():
        symbol = Company.objects.get(id=row[0])
        vlist = StockPropertyValue.objects.filter(symbol=symbol)
        vmap = {}
        for value in vlist:
            vmap[value.stock_property_id] = value.value

        values = []
        for header in headers:
            if vmap.has_key(header.id):
                values.append(vmap[header.id])
            else:
                values.append(None)

        result_list.append(Result(symbol, values))

    return headers, result_list
```

What is done here, is to load a list of headers, which is the list of stock properties that the user selected to see. Then for each of the company ids (the for loop) I construct a *Result* object, using the *Company*, and the list of values for the stock properties that the user is interested in. Finally I return the list of headers for use by the view, and the list of results.

Adding results to GUI

Now when I have the results for the user's Query. What I need to do, is to present it in the GUI. For this I use the same approach as I did earlier when adding criteria to the GUI. I create the search query as a AJAX request, and put the results into the *id='results'* in the search page. I still need to create the view and the template, but I am using the SearchForm from earlier.

Extract from views.py – getresult method.

```
def getresult(request):
    form = forms.SearchForm(request.POST)
    form.find_minmax_criteria(request.POST)
```

```
    if form.is_valid():

        headers, results = search.query(form.to_criteria(), form.cleaned_data['sector'],
form.cleaned_data['exchange'], form.cleaned_data['show_result'])

        # show result in response
        return render_to_response('search/result.html', {
            'headers': headers,
            'results': results
        })

    else:
        # show error message in response
        return render_to_response('search/result-error.html', {
            'message': 'Please enter details correctly.',
            'form': form
        })
```

The form is loaded from the POST request values of the HTTP request. If the form is valid, then query is performed, and we send to the *search/result.html* template, including the headers and results from the query. If there is any error, then we show an error message.

Template result.html – the search result template.

```
<table>
  <tr>
    <th>Company</th>
    {% for header in headers %}
    <th>{{ header.name }}</th>
    {% endfor %}
  </tr>
  {% for result in results %}
  <tr>
    <td>{{ result.symbol.symbol }} <img src="/site_media/
{{ result.symbol.exchange }}.png" /><br />
      {{ result.symbol.name }}
    </td>
    {% for value in result.values %}
    <td>{{ value }}</td>
    {% endfor %}
  </tr>
  {% endfor %}
</table>
```

There are three for loops inside the template, first one is to show a dynamic list of the headers, which is the stock property names. The second loop is to show all the results. And inside this loop, there is a loop to show the values for each result.

Historical data in the database

My final implementation task was to add the historical data to the database. By design it was luckily a simple task. The most included adding a model to the already created models, and having that one saved also.

Extract from models.py – StockPropertyValueHistory class

```
class StockPropertyValueHistory(models.Model):
    """
    Stock property value history holds the historical values for a given
    stock property value.  It is related as  a 1:M to a stockproperty value.
    """
```

```
    current_value = models.ForeignKey(StockPropertyValue)

    historical_value = models.DecimalField(max_digits=19, decimal_places=5)
    historical_date = models.DateField()
```

Like with other models the stock propery value history also uses the Django model.Model framework to declare attributes. What we notice is that we can use date fields in database, and also how we specify the foreignkey to another table. As we recall from earlier, Django can generate SQL schema from this model, and that I did for this model too.

Last was then to modify the extractor so it saves in a different way.

Extract from extractor.py – part of extract method.

```
    # get historical values
    hist_values = StockPropertyValueHistory.objects.filter(current_value =
sp_value).order_by('-historical_date')

    # there is no historical value yet, or the first (sorted, so newest) is
    # not the same as the value we converted.
    if len(hist_values) == 0 or hist_values[0].historical_value != converted:
        hist_value = StockPropertyValueHistory(current_value=sp_value,
                                               historical_value=converted,
                                               historical_date=datetime.date.today()
)
        hist_value.save()
```

Here I am finding all the *StockPropertyValueHistory* objects for a specific *StockPropertyValue sp_value* (see the design part for how it is modeled), and I order it by historical_date in reverse order, meaning I get the latest value as the first item. Then I check if there are any historical values, and if there is, then I check if the latest historical value is the same as what I just extracted. If they are different, I create a new historical value in and save it in the database using the *save()* method on the model object.

Implementation conclusions

For my conclusions on the implementation part, it has been many things to learn. It has been hard to find the documentation to times, and also since the tool support is not so good for Python and Django, I have often had to ask Freyr for some hints on what to do. But, I have learned, and I think that some of the parts I implemented early in the project, already now I can see how it could be done smarter. For example the search module, there are interfaces in Django on how to create custom queries so they integrate with the framework.

Also web programming in general has been quite a challenge. For example AJAX I don't know anything about, and still I do only have a very little knowledge about how it actually works, only by the help of libraries, it is possible to make it work easily. HTML and CSS design in general is also a challenge. I have a very simplistic design compared to other websites found on the Internet, but to create something beautiful is not easy without big work.

Testing

For testing, there are two parts that we have learned about in the classes. One is the user acceptance test, which is performed by customer, as a black box test, and then there is the module tests, which are performed by developers during development, usually as white box testing.

By using XP, I am getting good experience with using both. In iteration planning, each user story

will be as one user acceptance test, and each task created from the user story will be one module test.

Module testing

I have tried to follow XP by creating tests before I implement the code. While it is not possible for all the tasks (some tasks that are not product related, it is hard to create test), then most tasks it is possible to create a test, and then implement the feature afterwards.

I think this approach is very good, by creating the test, I am forced to think about how the functionality must be, and same time, the test functions as a kind of documentation for how my program works.

Django has a framework, where I create tests inside each application, and then execute one script, that will run all the tests. Below I show the example of one such test, which is the test for task no. 4:

Test for task 4, test existence and CRUD operations on a model.

```python
# Test for Task number 4.
class StockPropertyValueTest(unittest.TestCase):

    def testInstantiation(self):
        from models import StockPropertyValue
        o=StockPropertyValue()

    def testCrud(self):
        from models import StockPropertyValue, StockProperty

        # check that database is empty
        ol = StockPropertyValue.objects.filter(symbol="tdc.co")
        self.assertEquals(0, len(ol))

                # create stock property
        s = StockProperty(name="test", url="http://google.com",xml_path="test path",\
                        convert_expression="convert")
        s.save()
       # create one object and save to datbase
        o = StockPropertyValue(symbol="tdc.co", value=6, stock_property=s)
        o.save()

        # show the id just for ourself
        print o.id

        # read from database
        o2 = StockPropertyValue.objects.get(id=o.id)
        self.assertEquals(6, o2.value)
        self.assertEquals("tdc.co", o2.symbol)

        # update object and save to datbase
        o2.symbol = "test2"
        o2.value = 9
        o2.save()
        # compare objects
        o3 = StockPropertyValue.objects.get(id=o2.id)
        self.assertEquals(o.id, o3.id)
        self.assertEquals(9, o3.value)
        self.assertEquals("test2",o3.symbol)

        # delete
        o3.delete()

                # verify its gone
```
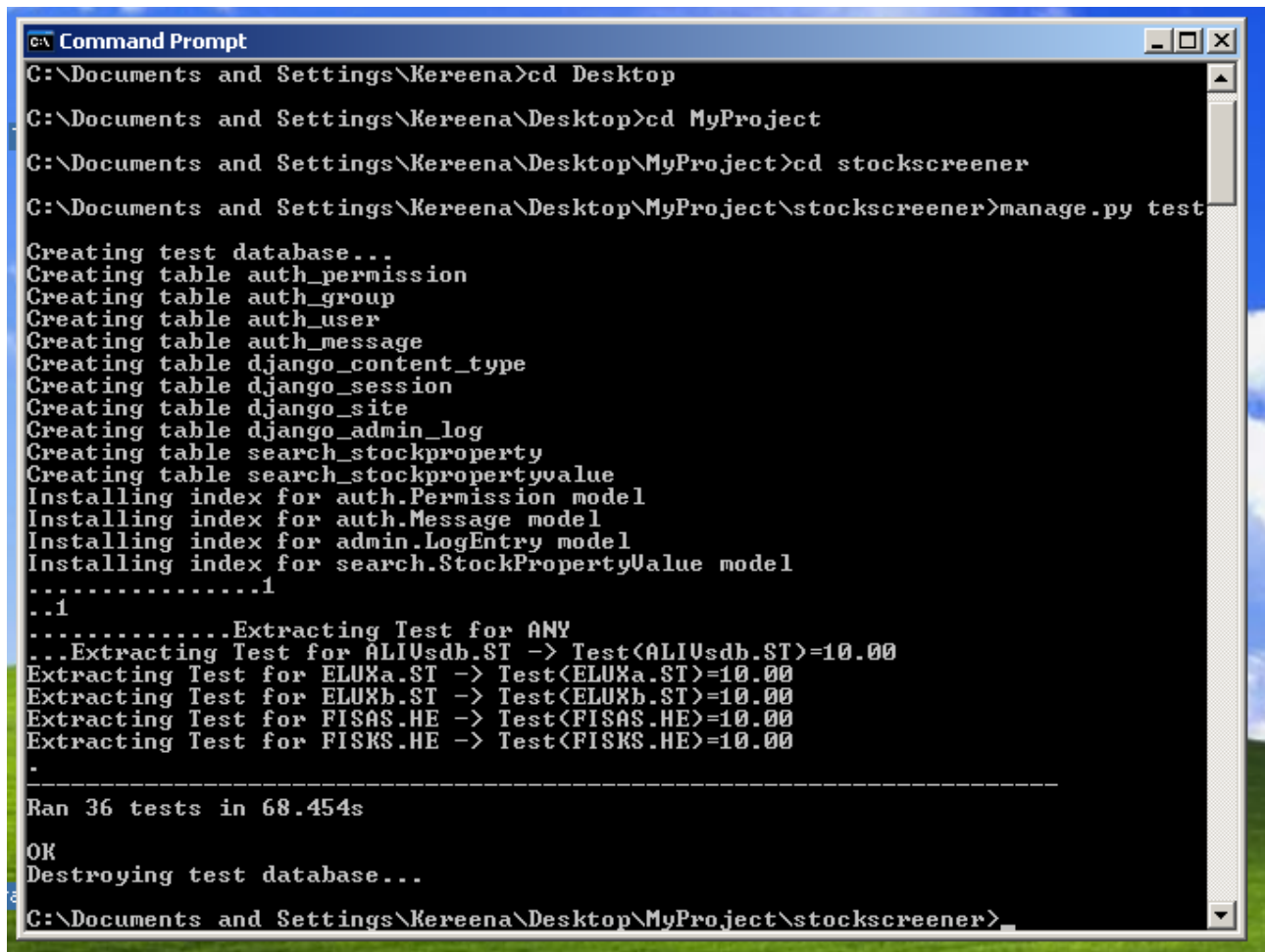
```
        ol = StockPropertyValue.objects.filter(symbol="test2")
        self.assertEquals(0, len(ol))
```

After each task, I run the tests, so I can see if anything has become broken in other parts of the application. Below is output of when I run the tests:

```
Command Prompt                                                          _ | □ | X |

C:\Documents and Settings\Kereena>cd Desktop

C:\Documents and Settings\Kereena\Desktop>cd MyProject

C:\Documents and Settings\Kereena\Desktop\MyProject>cd stockscreener

C:\Documents and Settings\Kereena\Desktop\MyProject\stockscreener>manage.py test

Creating test database...
Creating table auth_permission
Creating table auth_group
Creating table auth_user
Creating table auth_message
Creating table django_content_type
Creating table django_session
Creating table django_site
Creating table django_admin_log
Creating table search_stockproperty
Creating table search_stockpropertyvalue
Installing index for auth.Permission model
Installing index for auth.Message model
Installing index for admin.LogEntry model
Installing index for search.StockPropertyValue model
................1
..1
.............Extracting Test for ANY
...Extracting Test for ALIUsdb.ST -> Test(ALIUsdb.ST)=10.00
Extracting Test for ELUXa.ST -> Test(ELUXa.ST)=10.00
Extracting Test for ELUXb.ST -> Test(ELUXb.ST)=10.00
Extracting Test for FISAS.HE -> Test(FISAS.HE)=10.00
Extracting Test for FISKS.HE -> Test(FISKS.HE)=10.00
.
----------------------------------------------------------------------
Ran 36 tests in 68.454s

OK
Destroying test database...

C:\Documents and Settings\Kereena\Desktop\MyProject\stockscreener>_
```

As the number of tasks grows, also the number of tests to execute grows. For the full tests, see **Appendix-A**, tests.py source code.

User acceptance testing

Here I have transformed each of the user stories into a test that can be performed by customer when I deliver the functionality. In my table I have shown the created tests, and same time, I have used the table for collecting the information about when user accepted the test, and comments if it was rejected or accepted.

Uno	Test description	Notes	Delivered	Accepted
1	Install the software according to installation document.	Jens: N/A Soren: N/A	19-9-2008	20-9-2008

	Open url in browser: http://yani.dk:7400/admin/search/stockproperty/ (Replace yani.dk with Freyr's own IP) Verify all four required fields are available.			
2	Install the software according to installation document. Open url in browser: http://yani.dk:7400/admin/search/stockproperty/ (Replace yani.dk with Freyr's own IP) Verify P/E Ratio, Shares Out , Div and Yield are available and correct. In console execute the following: manage.py importone TDC.CO Verify that the extracted data is matching the data for TDC.CO in reuters website.	Jens: I would like some more properties to be extracted. Soren: I think some error handling is missing.	19-9-2008	20-9-2008
3	Install the software according to installation document. 1. Open url in browser: http://yani.dk:7400/distrgraph/4/ - graph with data, verify the graph is there and check if graph is suitable for test data. http://yani.dk:7400/distrgraph/99/ - graph without data, verify the graph shows "No data available". (Replace yani.dk with Freyr's own IP)	Jens: This looks good. Soren: N/A	19-9-2008	20-9-2008
4	Install the software according to installation document. In console execute the following: manage.py importall 9999 Verify that the extracted data is matching the data in reuters website.	Jens: What is the 9999 number for? (Yani: Its for specifying number to import, eg 10 when testing)	26-9-2008	26-9-2008

		Soren: N/A		
6	Install the software according to installation document. Open url in browser: http://yani.dk:7400/search/ Validate that the data in the search form matches the data available in the database. Validate that selecting a criteria in the "Add criteria" dropdown adds the criteria to the list of criteria. Validate that removing a criteria by clicking the red "stop sign" button works. Validate that the min and max values pre-filled for a selected criteria are correct as the min and max values in the database.	Jens: It looks good, smart with the AJAX. Soren: Maybe the new criterias could be added last instead of top?	26-9-2008	26-9-2008
7	Install the software according to installation document. Open url in browser: http://yani.dk:7400/search/ Validate that the "Search Now" button shows expected companies given some criterias.	Jens: There is no ordering on the search results. It would be good with that. It is fine that it only shows 25 Soren: N/A	24-10-2008	24-10-2008
8	The procedure is the same as the user acceptance test for Uno 4. After importing, go to the administration interface, and check that the values in the Stock Property Value Histories table are matching with the values in the Stock Property Values table.	Jens: Looking forward to API that can provide these values. Soren: Works as expected.	24-10-2008	24-10-2008

Conclusion

By this project establishment, I have clear idea about goals, requirements for this project and the possible risks. By strategy analysis I could see my strengths and weaknesses to work with this project or to find another and select the suitable process model to this project. By choosing XP process model, I am more certain by these tasks improvement by iterations. Company already can work with finished tasks instead of waiting for other tasks to complete. By time and cost estimation and review, I controlled their variation, because Time is money :-)

By working with new tools in this project, I got better knowledge by comparison between these new tools and Microsoft tools what I learned earlier in school. And also I got confidence that, I am able to work with new tools in real world.

Working with Freyr, I got real pair programming experience and also I develop the communication skills. This project gives me real job experience in real world with real problems. Working with XP is more exciting to me because, I am getting new ideas and knowledge how to improve better than earlier iterations. However, my project is still limited to the 10 weeks, so there are still many further developments that I can take to make it ready for production use.

I think that Freyr is completely happy about my work, and also they are suggesting some new user stories that I can work on, if I like to work with their company in the future. I am not sure they will use all parts of my project, for example the GUI design is different from theirs, but they are saying that they want to use at least the extracting parts.

All in all, I think my project is a success, and I hope it is sufficient for showing that I am eligible for passing the final semester at Datamatiker education.

Acknowledgments

I would like to thank my supervisor Jens Toft Madsen for good and helpful input during the meetings we had while working on the project. Also I would like to thank Jens from Freyr for giving helpful input for specifying my project and helping with Django questions. Finally, I would like to thank my husband Søren for trying to answer my questions, and for his patience while using him to practice pair programming.