Project plan and evaluation

Term 3 project: Business Intelligence system

AUTHOR: PATRIK BÄCKSTRÖM

VERSION: 1.0

Table of contents

1. Introduction

This document contains information about project management, and how the initial sprint plan has been carried out. It summarises and evaluates previously delivered *SCRUM PM* and *Design Document*.

1.1. Purpose

The purpose of this document is to introduce the reader to the underlying attitude of project management, individual roles and responsibilities, and evaluations of initial agreements, sprint plan, design and architecture ideas, and quality attributes. It resonates the attitude and atmosphere of individual, as well as group contributions.

2. Work approach

2.1. Project management attitude

Management and group work was settled to follow SCRUM. Considering this project, I, as project manager, was also given the role as SCRUM Master.

It was commonly understood and agreed upon, that apart from each individual's role specific responsibilities, we expected, and demanded a will of keeping up with existing sprint plan, daily SCRUM, and appointed meetings. Contribution should always be of equal interest.

2.2. Meetings

To be consistent with previous project, meets was set to Mondays, 10 o'clock, with re-scheduling possible when lectures occurred at set time. Meetings were intended to include all courses given during the term, thereof not scheduled on one of the days dedicated for the project (Tue-Thu). As discussed in our social contract, late arrivals are not accepted, and time should always be respected.

2.3. Sprint plan

According to our initial design document, sprint 2 through 4 was dedicated to back end and ETL development. From sprint 5 and forward, no precise planning was taken into consideration. The reason for that was that we did not know enough about ETL, noSQL, and Erlang in general, to supply satisfactory planning.

The following table illustrates the sprint plan carried out in the end of sprint 2.

Sprint	General task
2	Discuss requirements. Research noSQL databases, stock data sources, Erlang XML parsing, and in general how to fetch data in Erlang with HTTP-requests.
3	Create data fetchers and support with overall software design. Integrate ETL with CouchDB as much as possible. Discuss general interface design and functions. Implement ticker symbol fetchers.
4	Iterate current ETL, software design, and UI Design. Leave room for catching up on data fetcher and symbols fetchers. Develop PHP interface (middle layer).
5	Iterate ETL and software design. Begin front end development, interfaces specifically. Supply AT LEAST one view in all interfaces. Set up server to host project externally.
6	Iterate ETL (beginning interface design will reveal necessary additions). Implement more views in all interfaces.
7	Iterate ETL if necessary. Make sure ETL is ready to be released, and put on server. Extend interfaces to handle all views.
8	Finalise interfaces and documentation.

The above sprint plan was in general satisfactory carried out. A deeper analysis is found in section *3.3 Sprint plan evaluation*.

2.4. Quality and technical goal

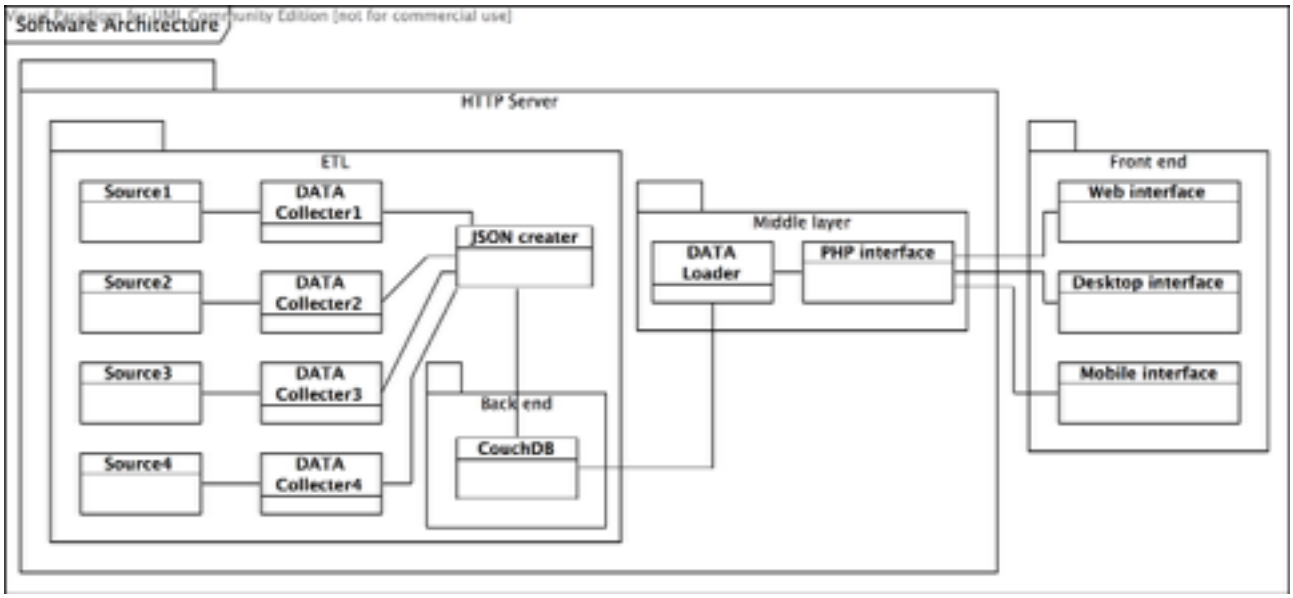
We agreed to be strict on coding convention, and in the startup phase it was up to our designer to research conventions for Erlang, and further more to verify that it was followed. Other conventions was to be considered depending on which languages our clients were to be developed in.

PivotalTracker was to be used for backlogging, whereas Bitbucket.org was to be used to host a common repository for documentation, final code, and working branches. We were to use Bitbucket's built in issue-report service to keep track of bugs etc.

2.5. Roles and responsibilities

Project Manager	Patrik Bäckström
Architect	Ludvig Gee
Designer	John Burchell
Quality Manager	Michael Thompson
Interface designer	Carl Berglund
Technical expert	Nico Boh

2.6. Initial architecture and design



3. Evaluation

3.1. Meetings

Meetings were re-scheduled to Tuesdays to suit over all program schedule better. It was also discussed and approved that meetings should be aimed only towards project matters.

3.2. Sprint plan evaluation

Many changes were made to the initial sprint plan on the fly. Setback is based on how long it took us to solve the mentioned task. It does not state that the project as a whole got delayed that time. Without asking for reasons, here are our major setbacks:

Sprint 2

Event: Erlang XML parsing was put on hold due to lack of knowledge.
Setback: None at this point.
Solution: We decided to try further into the project, if RSS was proven necessary to store in the database.

Sprint 3

Event: All data fetchers were not implemented.
Setback: Three weeks.
Solution: One additional fetcher was implemented in sprint 3. The last fetcher was re-scheduled to sprint 5. Two fetchers were enough at the moment and with a lot of similarities to the other fetchers, the last one would not take long to implement.

Sprint 4

Event: Existing ETL was abandoned.
Setback: Incredibly hard to estimate, but all in all from four to eight weeks, depending on how one looks at it.
Solution: A new ETL with huge quality, reliability, performance, and fault tolerance improvements was developed, making us feel very secure about quality in all parts of the project. This truly gave us much better understanding of Erlang/OTP.

Sprint 5

Event: Development of interfaces did not begin as agreed.
Setback: Four to six weeks. Also affecting further ETL development.
Solution: Scheduling many, many, more hours of work in sprint 6, 7 and 8.

Event: ETL version 2 made us thinking of quality a lot more. Extra features (i.e. batcher, logger, and system initialiser) took more effort to implement.
Setback: One to two weeks.
Solution: Since interface development had not got started, two group members had enough time to implement requested functionality.

Sprint 8

Event: PPP Examination took place. This was badly planned and took all members one week to accomplish.
Setback: One week (or rather six?).

Solution: Abandoning some minor requirements, affecting interfaces to have less functionality.

3.3. Quality and technical successes

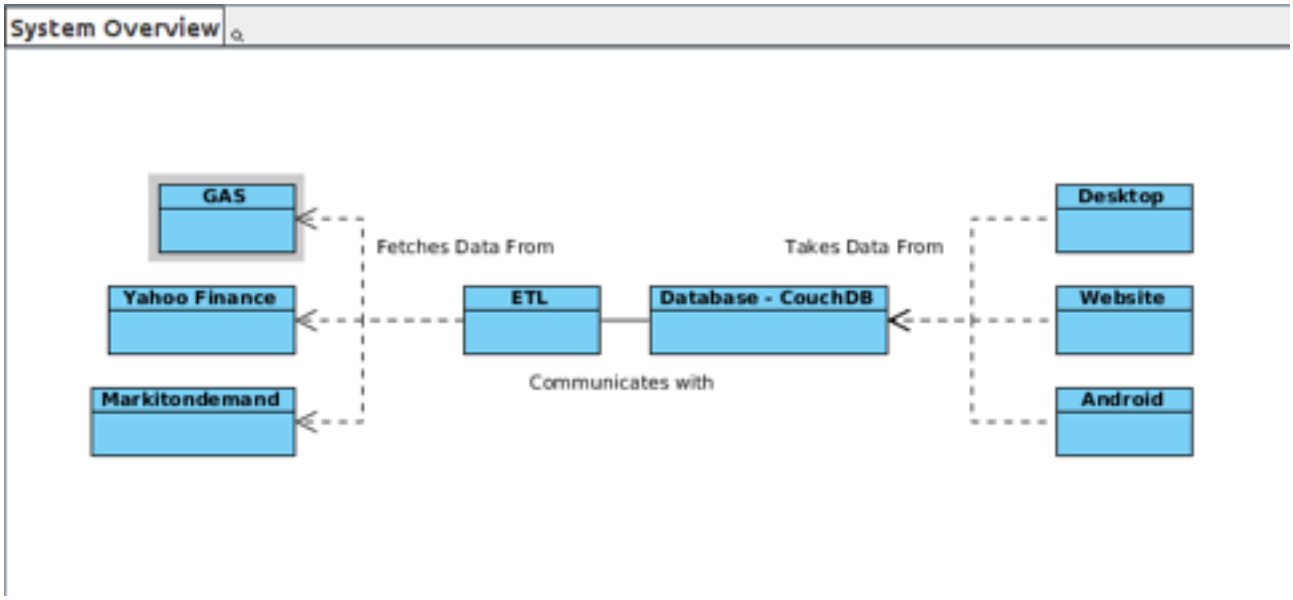
Code convention documents were created as needed. Erlang conventions were set and documented as soon as we felt experienced enough to understand the meaning of the convention.

Interfaces were developed for three platforms; Android, C#, and web. Coding conventions follows the same well-known standards available from respective supplier. Conventions more meaningful to us are further elaborated in *Software Design Document, section 5.1*.

PivotalTracker was used for setting up tasks, but to be honest, poorly used. We usually found tasks unnecessary to add after the subjects were discussed.

BitBucket has served as online repository and was used on daily basis. All final code and documents etc are present. Branches were diligently used for safer and less error prone work.

3.4. Final architecture and design



The most significant change is that the over all HTTP server was abstracted away. The reason for this is that the server we use is a typical stand alone server hosted by a member of Group Adapa. It holds no unique advantages from other FTP servers, and runs “as it is”, and does not need to be considered a vital part of *development*.

Secondly, the previously planned middle layer, was also abstracted away because all interfaces have the possibility to request data straight from the database through CouchDB’s RESTful interface, thus making that our middle layer.

The planned specific data loader was also replaced by CouchDB. The fundamental idea was to use a data loader to make sure that correct data was returned when requested. Realising that we could perform the same validation whilst fetching data, made such a module redundant, bringing a more complex usage to all interfaces.

Previously unmentioned in the initial project plan, is how our ETL implements Erlang's OTP standard. That gave us enormous advantages in how errors, failures, and software health was indicated. The amount of work needed to redo the entire ETL was huge, but well worth it. Since we released and started the final version, no unexpected errors* have been logged when running it 24/7!

** expected errors exists. For example, badly formatted returned JSON shall always be considered an error, thus be logged.*