Summary Report for MyCourses by Scetris

Who we are

A short introduction stating names (with a picture) and skill level.

We are a team of five undergraduate students of the Free University of Berlin. Even though we are all in the same year our knowledge regarding the creation of software and the used technologies varied widely. We were however all inexperienced in project management and none of us had been part of a software development process before. This said we will introduce ourselves with names and a picture so you can get an idea.

….

Intended and actual process

A description of how the skill level and composition of our team influenced our choice of development process and how we modified that process to fit our need, giving a rationale for each.

As mentioned before our team lacked experience in software engineering and we chose to follow a mostly agile software process as we were certain that we would learn a lot along the way and our first ideas and predictions were hardly precise. Choosing an agile approach we hoped to adapt easily to the things we would learn and thus rapidly improve our process and our design. While a strict agile approach was infeasible with all of us spending most of our time on courses at university we used our own ideas roughly based on agile software engineering. Our approach consisted of heavy usage of technology to ease communications and organization, regular meetings with the whole team, working alone and often in small teams without fixed times. All of these will be explained in more detail now.

We decided to use a trac system providing us with tickets and a wiki to coordinate the assignment of tasks and gather information about progress and possible problems. To ensure consistency in the code and gain backups we used subversion. For general communications a mailing-list was set up and heavily used.

We held regular meeting with the whole team to discuss what has been achieved, what could be improved and how the time until the next meeting will be spend. These meetings addressed mostly organizational issues but design and implementation were also discussed. From the end of May until the beginning of October meetings were held on a two-weeks basis. With the beginning of our third iteration in October we picked up the frequency and met weekly. This is mostly due to the fact that productivity increased a great deal around September and with the beginning of our third iteration increased coordination of efforts was necessary because the parts of our systems were ready for integration with each other.

Much of the work was done alone due to the fact that we could hardly find suitable appointments and the experience that productivity slipped with more people present. Still a significant amount of development was done in small teams. This was reasonable as we tried to avoid having single experts, because of the risks this poses. In particular we used pair-programming to implement the scheduler. This was done for two reasons, firstly to ensure a high quality of the code and secondly to have two people equally familiar with it.

Problem Statement

Score Project description. Point out the areas that were unclear to us and how we gathered information to clarify these areas.

Organizations like Universities and Schools are required to allocate courses to the given resources in a sensible way. As the amount of resources and courses grows large, it gets increasingly costly to do the allocation manually. Furthermore the task is repeated quite often, so a system to ease the process is very useful. The ideal scenario is a program that automatically finds an optimal solution. However the problem is NP-hard and thus a computer-aided scheduling process is the focus. As the interests of many employees and students at the universities are affected by the result, they should be taken into consideration – the students that is – or even be allowed to participate in the process.

Our Design and Decisions

Short introduction stating that the different categories of design will be described and a rationale for each will be given.

Many areas of our project required decisions and a rationale behind it. These decisions are sorted by area below and each decision is explained shortly. In retrospect some of these decisions were less than optimal and a description of how these influenced our project and what we learned will be given.

Technologies

Why we used the technologies we did and why we did not use some others

Programming Language

- AspectJ, as we were all familiar with Java and wanted to have aspects, which is especially useful to supplement auto-generated code. Regarding the amount of time spent to learn new technologies and the fact that we did not have much of a time buffer, this was the right decision. Using AspectJ did not force anyone in our team to learn a new language but still provided powerful features that were utilized by some where useful. Especially considering the huge amount of auto-generated code, AspectJ was a great choice as it allowed us to plug in functionality in some auto-generated classes without changing the other auto-generated classes and also allowed do this in a most simple way.

Database Management System

- Postgres, as we wanted it to be open source and had already worked with Postgres.

Object-Relational Mapping

- No existing solution but our own as we felt that existing solutions would take a similar amount of time to get used to and we would learn far more implementing the ORM ourselves. In regard to future work on the program, whether for maintenance or extension, it might prove beneficial to replace this solution with a more widely used tool, such as Hibernate. This would improve the process of understanding our software for people familiar with said tool. However we are satisfied with our own solution and we will stick with it, as it would be far too time-consuming to refactor this part of our project and we do not plan to work extensively with the software later on.

Web-Framework

Why we decided to use our own framework and how it is implemented. What the advantages are.

We decided to implement our own web-framework as we wanted an easy integration of web-application and scheduling-algorithm. Thus a web-framework that uses the same

language as our scheduling-algorithm was written to simplify the most common tasks. As well as getting a framework that fits our needs nicely, the benefit for us personally, which comes as knowledge was part of the reasoning against using an existing framework. We assume here that the knowledge of general principles is more beneficial to us than knowledge of tools.

Meta

Why a framework for common tasks has been created and how it affected our development process.

As many tasks regarding the functionality of our web-interface require similar methods and should be implemented in a unified way to ease understanding, a framework for the most common tasks was written. It is called meta and is completely separated from the rest of our project, thus it can and will be used after we finished this project. So not only did it ease the development process for this project but also it will pay of even more in the long run.

Scheduler-Algorithm

Why we have chosen a genetic algorithm for our scheduler and how we tested and improved performance.

A genetic algorithm was chosen to be the foundation of our Scheduler-Algorithm for the following reasons. We felt that why it is clear what general properties the result of a successful scheduling process must have it is most unclear what exactly is a good solution to a specific scenario. So an algorithm that is able to operate without being determined but will finish at some point if a solution is possible was necessary. Genetic algorithms seemed a fair choice and we found an example that we rebuild to get a more precise idea of how it works. After building this prototype we decided to stick with a genetic algorithm as the results were satisfying. Also we needed the algorithm to be adaptable to both one single monolithic scheduling and a collaborative mode. This means that it is necessary for the algorithm to scale nicely with different sets of courses and constraints. We believe that this can easily be achieved with genetic algorithms as these have a few parameters to influence the behaviour anyway and these should suffice. And it also means that while the monolithic scheduling needs the full functionality of our scheduler, this is not the case for the collaborative mode. As almost all functionality needed for collaboration is needed for the other mode as well, it mostly only takes a good modularized design to easily provide both modes.

The basic scheduler-algorithm was implemented in our second iteration and was refactored to use a greedy-algorithm to generate the first population. The decision to add the greedy-algorithm was based on a paper that showed a performance improvement through utilization of a greedy-algorithm.

The button

Description of our product. Features and the concept behind the UI. Possible improvements and add-ons.

MyCourses is able to create a semi-automatic scheduling for programs at universities or schools. It can also provide a scheduling that was run fully-automatic but the recommended usage is to have MyCourses create preliminary versions of a scheduling. These will then be improved by the users via the functionality for collaborative scheduling also provided by MyCourses. This process is best used iterative through several cycles of automatic assignment and manual improvement. To get better results from the automated scheduling users can define requirements and assign those to courses or resources. A

requirement that is assigned to a course is regarded as a constraint that either must be met in case of a so called hard-constraint or simply improves the rating of a scheduling in case of a soft-constraint.

In Furthermore MyCourses provides functionality to

- enrol in courses
- view your own schedule
- view schedule for a room
- import and export data
- create, read, delete and modify data related to schedules

Timeline

An outline of what happened when and where we kept the plan and where not. Possibly a graphical presentation. Must emphasize the fact that the plan evolved on the go.

We started work on our project in April but that was almost exclusively organizational, i.e. getting to know each other, deciding which professor we would ask to be our contact person at the university. During this time no real work was done and thus it was called iteration zero.

The first real iteration started on June 20 and comprised of an initial requirement elicitation, design of our software and prototyping of the scheduler. During most of this time the academic term was still running and only a fraction of our time was devoted to the project. This iteration ended on September 9 and was evaluated the day before. Much of the time spent in this iteration was still focused on gaining orientation. Still it was necessary for us to take this time and find a suitable approach to the work ahead.

The second iteration thus started on September 10 and was devoted to producing the scheduling-algorithm, the web-interface and our ORM as well as weave and meta. Work cycles were short and meetings were held every few days, including a weekend devoted to coding. From this point on our iterations lasted about one month each and the focus was on implementing and testing. Requirements and design were refactored whenever we felt the need to due to new insights or problems. This iteration ended on October 4 and was evaluated the same day.

The third iteration accordingly started on October 5 but most work was halted until October 16 due to exams and all-day courses at university. Beginning with October 26 weekly two-hour meetings were held at our university in which everyone reported achievements and problems of the last week, as well as set goals for the next week. These weekly meetings have proven to be a good way of coordinating work as well as motivate each other, so they were conducted for the remainder of the project. The main goal and achievement of this iteration was the integration of the scheduler with the web-interface and the connection to the database. Due to this employment of parts of our software many issues were found and fixed. Accordingly we shifted our focus further towards unit-testing and started using a tool to measure code coverage, that is how much of our code is actually being executed as part of a unit-test. At the same time more functionality was added to the web-interface. This iteration ended on November 1 and was evaluated the day before.

The fourth iteration started on November 2 and was primarily focused on the improvement of our web-interface, adding functionality and a unified design. Some improvements to the scheduler-algorithm were made, most notably the introduction of a greedy-algorithm for

the set up of the initial population. Work on the Summary Report was also started in this iteration and the first version which was the basis for all further versions was written. The iteration ended on December 5.

What we learned

What each of us has learned technology wise and what we have learned about projects.

Conclusion

A short summary repeating the process, and the product.