Titelseite anpassen: Umsortieren der Bereiche FU <-> SCORE

FREE UNIVERSITY OF BERLIN

DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE
INSTITUTE OF COMPUTER SCIENCE

Summary Report

Project MyCourses

DAVID BIALIK, JULIAN FLEISCHER, HAGEN MAHNKE,
KONRAD REICHE, ANDRÉ ZOUFAHL

DECEMBER 19, 2010

Contents

Abstract vor das Inhaltsverzeichnis

Abstract

The abstract...

1 Who we are

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. Specifically none of us had been part of a software development process before. This said we will introduce ourselves with a picture and names.

Foto machen + Infos zu Personen, mit Alter + Role im Projekt

Figure 1: From left to right: David Bialik, Konrad Reiche, Hagen Mahnke, Julian Fleischer, Andre Zoufahl, Mister X

2 Development process

evaluate

In the following two sections our development process will be described and criticized. The first section is a description of our initial ideas and the rationale behind it. The second section describes how our development process actualy happened and where we altered it.

pro: einleitung zu jeder section, aber weniger eigenkritik, mehr analysierend

2.1 Intended development process

As mentioned before our team lacked experience in software engineering, thus we chose to follow a mostly agile software process. The reasoning behind this, is the expectation to learn a lot along the way, while our first ideas and predictions would be imprecise and sometimes inapplicable. Choosing an process oriented towards agility we hoped to detect problems fast and change our process and design in a timely manner. While any strict agile software development approach was infeasible, with all of us working only part-time and the lack of physical proximity, we decided to employ whichever ideas seemed appropriate. The lack of proximity might be surprising as we all study at the same university. It is a

konkret ein modell nennen, hier spiralmodell, da wir das auch machen wollten, wir hatten starren blick und erhofften struktur

Bild von spiralmodell
(evtl) mit farbiger hervorhebung der
Iterationen in der spirale

valid claim nonetheless as our schedules made it near impossible for us to find common times for project related work. So most work was done isolated either through time or space or both. The choice of useful techniques was further limited by the fact that the amount and distribution of working hours were highly variable. Given our readiness to redefine our process, we expected it to change but failed to define criteria for monitoring the state of the process.

We are aware of the fact that some essential components of agile processes were not present in our project. Neither did we have a customer representative directly available, nor did we treat our contact persons as customers. In retrospect this perspective would likely have led to the implementation of key features much earlier and improved motivation. We also did not go through a full development cycle in each iteration. Accordingly we regard our process as being oriented towards agility and not as an agile software process in the true sense.

absatz ist schon actual process, muss also eine subsection weiter

Key components of our process were

- holding regular meetings for coordination
- intensive communication via e-mail
- evaluation of the process whenever we felt something was amiss
- employing a ticket system to define and monitor tasks

Substantivierung der Listenpunkte

2.2 Actual development process

im nachhinein wurde uns klar spiralmodell ist zu starr für das was wir machen, braucht mehr erfahrung, naives herangehen an terminsetzung, zu lange iteration

As we expected some of our initial ideas were unsuitable or even obstructive. The following paragraphs will discuss where we adhered to our initial plan and where we did not, as well as criticize our decisions or lack thereof.

As intended we held regular meetings 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. Within our third iteration in October we picked up the frequency and met weekly. This change is due to integration of parts of our systems with each other, increasing necessity for coordination. We kept meeting weekly thereafter as we found the interval to be fitting our work-cycles well. Due to our workload and our need for orientation at the start we doubt that a weekly cycle would have been appropriate right away. However we could have benefited from starting to meet weekly in August. At that time we often met several days in a row but used it for coding and did neither define nor evaluate tasks clearly. All the time we invested in August and September might have been used more efficient, had we taken the time to review each week.

E-mail was our main medium of communication throughout almost the whole process. As mentioned earlier the times at which we worked on the project differed strongly and an asynchronous communication was a natural choice. A mailing-list was used for organizational issues as well as for artifact related topics. All e-mails were sent to the list even if they were aimed at a single or few receivers. The assumption was that reading e-mails on all the topics would help distributing knowledge to the whole team. On the other hand the high amount of e-mails posed a significant amount of work if read thoroughly. One might also argue that reading isolated informations about topics does not work very well for passive knowledge transfer, so the positive effect might be even smaller. Another problem of our communication was the tendency to mix several unrelated messages into a single e-mail. As a result everyone was forced to read every mail at least fleetingly. Even worse information could sometimes not be found a few weeks later. A clear separation was attempted several times but we failed to adhere to it.

We did evaluate our iterations at their end and attempted to find weaknesses as well as ways to improve upon them. Positive aspects were discussed as well to gain a more balanced perspective of how we were doing. However success of our evaluations has been moderate as problems when identified were tackled ineffectively. On the other we did improve upon our process and the execution of it. We just did it whenever someone, often a group of people, noticed something amiss and informed the others. In all cases a discussion ensued and ended in decisions that were far better respected than the ones obtained during official evaluation. We assume this is due to the swift dealing with real problems, leaving only minor or even purely perception based problems for evaluation. We kept evaluating nonetheless as it was a good opportunity for giving feedback.

We decided to use a trac system providing us with tickets and a wiki. Both were used to coordinate the assignment of tasks and gather information about progress and possible problems. The usage of the ticket system was minimal in the beginning and only picked up later, leading to a conducive usage from October. This is also one of the examples where a problem was informally addressed by a member followed by a significant improvement. Prior to this point the problem had been minor, as almost all the work had been done with most of the team present.

2.3 Time-line

time line oder timeline

A short overview of the course our project took will be given in the next paragraphs. The goals for each pahse were defined only shortly before it's start.

phase

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.

iteration zero offiziell streichen,
mehr als pre-orgaphase

Timelines paragraphisieren

Farbiges Spiralmodell

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.

technologiefindung und nennen was wir genommen haben

The second iteration started on September 10 and was devoted to producing the scheduling-algorithm, the web-interface and our ORM as well as weave and bakery. 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.

iteration kürzer, damit schneller feedback, leichtes abweichen von spiralmodell

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.

evtl. konkretisieren, falls grafiken nicht aussagekräftig

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.

The fifth iteration started on December 6 and was mainly devoted to minor improvements to the scheduler and writing the beta-version of our summary report. One improvement to our ORM which made query-caching available drastically improved scheduling performance by a factor of 8.

3 Requirements

The next two sections are a overview of the problem and the resulting requirements. The first section will state the problem, while the second describes the specification we deduced from it.

3.1 Problem Statement

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 and thus a lot of work is spent for it. Often the courses and resources remain similar each term and much of the previous solution can be reused. Each solution to such a problem must satisfy a number of constraints. These constraints often are, but are not limited to:

- a room can only be booked once at a given time
- a lecturer can only lecture one course at a given time
- the different instances of a course must be held at different times

We called them hard-constraints.Furthermore a number of constraints exist that represent a preference. These constraints should be satisfied where possible but they need not be. These were called soft-constraints.

3.2 Requirement specification

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 allowed to participate in the process or at least be taken into consideration. As we desired more insight in how a scheduling process may look like, we asked an employee of our institute to give us a demonstration. The demonstration was conducted on the system used at our institute. While it did not reveal completely unknown aspects to us it gave us a better estimation of the importance of certain aspects. The requirements that struck us as most important are the following:

es ist egal, wie wir es herausgefunden haben, sondern was, siehe treffen mit employee

- it must be possible to have courses allocated automatically
- the automatic allocation must eventually satisfy all hard-constraints
- the automatic allocation should satisfy soft-constraints where possible
- a manual allocation must be possible before and after each automatic allocation
- the manual allocation must be suited for multi-user environments

Liste überarbeiten Substantivierung

konkretisieren

- information related to scheduling must be made available to the algorithm and certain users
- it must be possible to define what scheduling-related information is
- the information and results must be stored persistently
- the information and results must be presented to the users comfortably
- it must be possible to restrict access to information

skalierbarkeit
configurierbarkeit

These are the requirements that had the highest influence on our initial design and each subsequent change.

4 Architectural design: Our Design and Decisions

headline reduzieren auf nur den ersten teil

grouped by

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 while others turned out great and a description of how these influenced our project and what we learned will be given. The first section consists of a short overview of the system and a graphic representation. The second, consisting of several subsections will discuss the technology employed. The third section talks about weave, which is our web-framework. Finally the fourth section is about bakery, which is a collection of tools for automatic code generation.

kondensen der ersten 2-3 sätze

hinweis auf GUI in punkt application

4.1 Overview

Our system is best understood as a web-application providing the functionality to access the automatic solver and manipulate the data related to scheduling...

4.2 TECHNOLOGIES kriegt eine section unter AD, außer 4.2.4 - ORM bleibt in AD, als subsection

technologien kurzver overview , was wir genutzt haben, maybe point out FLOSS,

AD kriegt noch relationales modell

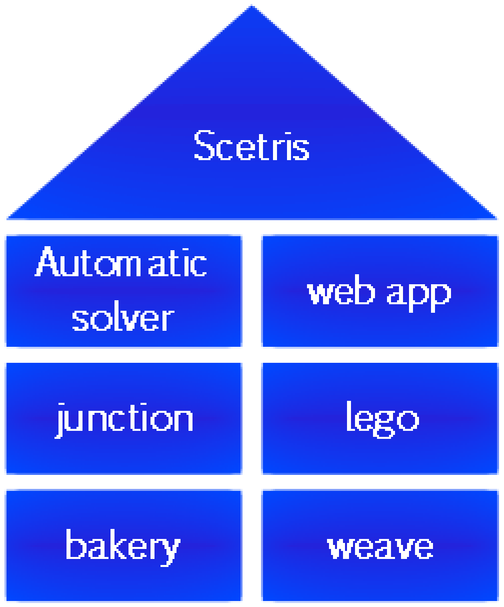


Figure 2: An overview of the components

4.2 Technologies

The following sections will discuss the technologies used in our project.

4.2.1 Tools

trac, ant, tomcat, eclipse, svn, postgres, (junit)

4.2.2 Programming languages

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

kommt raus

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.

4.2.3 fliegt raus, wird aber 'irgendwie' in technologies wiedergeboren

weitere sprachen für 4.2.2
JAVA
ASPECTJ
PLPGSQL
SHELLSCRIPTE

4.2.3 Database Management System

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

4.2.4 Object-Relational Mapping

auf julian warten :P

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.

4.3 Web-Framework

Web-Framework + Code-Generation + julians tolle grafik über unseres AD

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.

4.4 Bakery

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.

4.5 Scheduler Algorithm

The problem of course scheduling problem is based on the Job-Shop Scheduling [?]. The course scheduling problem is NP-hard and the optimal solution can be found by us using heuristic search algorithm. As a matter of fact this works only for simple cases. Todays course scheduling at universities are faced with complex constraints and as the input and requirements become more complex another

us using - using

klären was wir da wirklich haben ? meta heuristik, heuristik search, genetisch... ?

approach had to be considered. Based on some paper the genetic algorithm approach seemed to be promising and was chosen therefore.

satzinahlt umschreiben -> genetisch wird direkt am beispiel erklärt

As we will not go into detail about how genetic algorithm work in general, this will be explain on the application of course scheduling in concrete. The basic data-structure to model the course scheduling problem is a set of rooms with each having a list of time slots. The components of the scheduler divide into the following:

Greedy Setup generating initial population of possible solutions

Fitness function evaluating the constraint satisfaction by scoring it

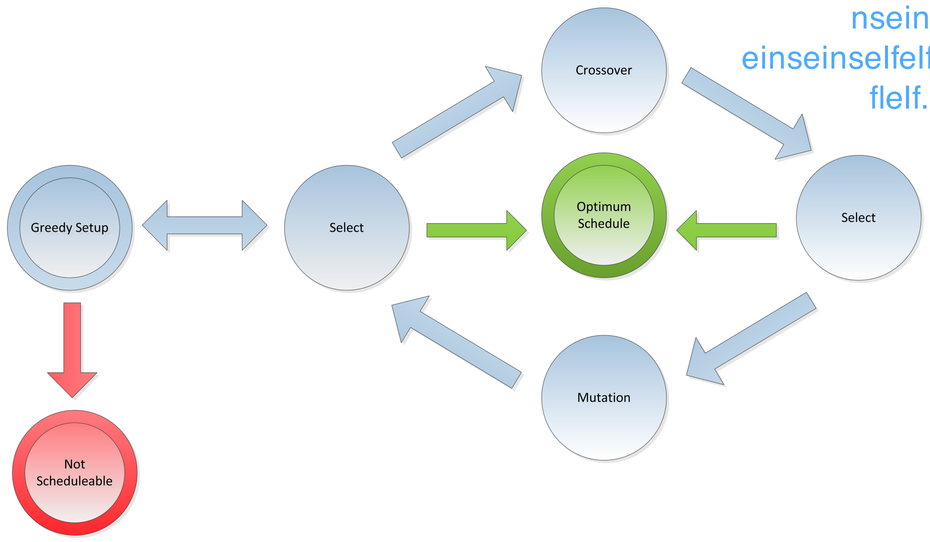
Crossover creating a new possible solution by mixing course starting time slots of two possible solutions

Mutation changing a possible solution by moving randomly chosen courses to new time slots.

an initial pop

Figure 3 illustrates the components interaction. At the start of each scheduling a initial population containing n possible schedule solutions is generated by *Greedy Setup*. Randomly chosen courses are placed into the room which fits the course constraints, for instance number of seats, the best. In this step the time slots are chosen at the earliest possible time slot and are there distributed along all days at the week. If, however, there is a constraint for a preferred room or a preferred time slot entered by the main lecturer this spot is chosen instead directly.

With the possible rise of finding a optimum schedule while generating the initial population each schedule is scored with the *Fitness function* and selected into a group of best schedules. The *Fitness function* is divided into two functions. A *Hard fitness function* calculates a score of the hard constraint satisfaction, this includes especially solving the room-time constraint and all constraints configured to be high priority constraints. The remaining constraints are scored by a *Soft fitness function*. When all constraints are satisfied an optimum schedule was found.



größere schrift !!!!!!!!!!!!!!!!!!!!!!!!!!!! 11111111111eeinseinseins einseinselfelfeflefelfle flelf.

Figure 3: Routine of the scheduler algorithm

Scheduling bigger input including complex constraints shows that a optimum schedule is not found in the phase of *Greedy Setup*. At this point the genetic algorithm is applied. *Crossover* and *Mutate* is executed on the present schedules. After each operation the new schedules are evaluated by the *Fitness function* until an optimum schedule was found or the scheduling is stopped by the user.

5 Implementation

abstract fehlt

subsections: scheduler database-access-layer web-framework stored-procedures

5.1 Scheduler

The scheduler was implemented as backend of our application and was therefore a core component. After researching possible implementation approaches we decided to implement the basic structure of the scheduler in pair programming. We expected this decision will result in a well-conceived design and basic implementation. The second reason was two have at least two team members being knowledgeable of this component.

pair programming kommt zu 2.2 actual dev process

A lot of effort was spent into the implementation of the data-structure modeling a possible schedule solution in order to apply the genetic algorithm in an efficient way. Genetic algorithms operation are based on a data-structure which enables them to be applied. *Crossover* and *Mutation* need properties which can be manipulated.

For these reasons a *Map* was used.

$$Course \mapsto (Room, TimeSlot)$$

Applying *Crossover* and *Mutation* means to take direct affect on this map. However this *Map* serves only as an access for the genetic algorithm. Behind it is the data-structure modeling the actual schedule. A *List* of rooms, each having a separate *List* of time slots. Further each time slot has a *List* of Courses as random factors of our algorithm rise possibility having two course at the same time in the same room. This data-structure is illustraed in the following figure:

6 Verification and Validation

cobertura
benchmark

6.1 JUnit

6.1.1 Scheduler Validitation

7 The Application

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

what has been achieved ?!

what is possible ?!

GUI

7.1 Usability

mycourses ?
unser produkt
heißt
SCETRIS

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

8 Outcomes and lessons learned

8.1 Outcomes

I dont know wether this should be located here or is intended to be somewhere else The web-application we have developed has some limitations due to different reasons *reasons to come*. It does not do this and that and another feature we should have implemented is not functional at the moment. The features should be easy to add, because they were considered during system design and no interface changes have to be made.

Beside our final application we also produced some side products which are worth to be mentioned. These include but are not limited to:

- junction
- bakery
- weave
- lego

analog zu 4.5 scheduler-algorithm

to be explained asap (as short as possible) by someone who knows how to do that

Additionally we produced a large amount of project- and other documentation. (Details here)

8.2 Lessons learned

termfindung besser

codecoverage

For all of us this was the first project of this size and duration. We took the chance to put into practice what we had learned in our software engineering course. Especially things that are not applicable for smaller projects. Things like (list of things).

We had to learn, that project-oriented approaches are hard to adhere to in an educational setting. Our team could not meet as often as claimed by nearly all development methods. Nevertheless we managed to establish a systematic meeting shedule.

At the beginning the proposed architecture of our software changed often and radical. Even with a given set of written down requirements. This was due to hard to understand requirements. There where different interpretations of the described requirements. We got it right after the first phase mainly because of (why?).

Our present team had not worked together previous to this project. We experienced that we are very different people with different approaches to problems. Near to the end of this project we are all friends and doing stuff together in our freetime. Everyone learned to get along with the originalities of the others. We have to highlight, that respect and honesty are very important in a team to lead to a productive development environment.

We had to undergo the experince, that a complex development environment has to be easy to set up. Otherwise it will consume a lot of time even after small changes to get ready to work again.

During the project we have a good chance to learn some new technologies List of things we learned stated by the team members:

- XML

9 Conclusion

A short summary repeating the process, and the product.

References