SCORE Report for MyCourses A Course Scheduling System UCSC Team

Ben Ross (bpross@ucsc.edu)
Erik Steggall (esteggal@ucsc.edu)
Justin Lazaro (jlazaro@ucsc.edu)
Sabba Petri (spetri@ucsc.edu)
Will Crawford (wacrawfo@ucsc.edu)

Contents

1 Executive Summary

The MyCourses scheduler system is an end-to-end scheduling system that automates most of the manual labor involved in scheduling university courses. We seek to provide universities an easy-to-use service that can help reduce the costs and complications involved in organizing an entire schedule.

Our team started in mid-September as a group of 5 engineering students taking a "Software Methodology" course. We chose to do a course scheduling system as our class project because we felt that the problem was interesting, and that we would learn a lot by writing a system to solve it.

The members of our team are Justin Lazaro, Will Crawford, Ben Ross, Erik Steggall, and Sabba Petri. Our team members have held a variety of roles in the technology industry such as IT, QA, Marketing and System Administration at top technology firms including Hewlett Packard, Riverbed Technologies, and Yahoo!.

2 Development Process

This instance of the MyCourses scheduling system was created as part of a class instructed by Professor Linda Werner here at UCSC. Participation in this class required that we approach design very seriously before we started coding. This was a new approach to us as most of our team was accustomed to developing through rapidly iterated prototypes.

After gathering the requirements from the SCORE project description, we immediately discovered that scheduling classes would likely be an NP-hard problem, meaning that there would be no way to find an optimal solution in polynomial time. In other words, it looked like a problem with no canonical "best answer." Much of our time in this phase was spent shopping around for a similar problem whose solution we could apply to our problem. We eventually found a solution in the form of a freely available course scheduling algorithm written by a Serbian software developer named Mladen Jankovic. It was written in C++, so we converted it into Python.

We started out planning our project using the Unified Process, an iterative and incremental software development model. The Unified Process is composed of four phases: inception, elaboration, construction and transition.

We lacked familiarity with the some of the technologies we planned on using for our project (Django, our Python web framework, in particular). Because of this, we spent much of our inception phase and elaboration phase learning about the functionality and capabilities of these technologies. This allowed us to generate more realistic design documents for our project.

Since this project was being done as part of a class, our instructor required several deliverables from us apart from the requirements of the SCORE project as part of a modified Unified Process. These were primarily presentations and the delivery of documents that we will discuss further in our Project Plan section.

3 Requirements: Problem Statement

Scheduling courses is a difficult task for many universities. Manually scheduling hundreds of courses can be very tricky due to limited classroom space, limited professor availability, and limited time to teach these classes.

We imagine four primary stakeholders for this task: Program Administrators, Program Managers, lecturers, and students. The Program Administrator will be responsible for determining a master list of courses, classrooms and time slots. The Program Managers will select which courses to offer in a given quarter and provide additional information about them such as class size and class location. The lecturers (or professors) will prefer to teach certain courses. They will also prefer to teach at certain times. The students will need to enroll in courses offer in a certain quarter.

3.1 Scenario

Below, we outline a common use-case scenario. This scenario was generated along with many others during our inception phase in order to help us

understand how users would interact with a system designed to solve this problem. This particular scenario involves a Program Administrator and several Program Managers.

1. The Program Administrator signs in to a browser-based system by supplying their credentials to the website.
2. Once logged in, the Program Administrator is presented with a panel illustrating their privileges - there will be links to pages that let the admin modify the database, add users, etc.
3. The Program Administrator can click on the database modification link, and will subsequently be presented with its contents in a tabular format.
4. Choosing to import data, the Program Administrator will need to provide the following:
   (a) A list of all classrooms and buildings available, including...
      i. Room capacity.
      ii. Subject preference. (e.g. Social Sciences belong in the Sociology building)
   (b) A list of all instruction time slots. (e.g. Can classes be taught on Saturday? When are normal instruction hours?)
   (c) A list of courses offered for every subject.
5. The Program Administrator can also manually enter information (e.g. Baskin Engineering $\rightarrow$ Room 156) or modify imported information. This can be done by clicking on the appropriate element, modifying the appropriate fields, and submitting the changes.
6. Once the information has been imported and reviewed, each Program Manager can determine which courses will be offered in the upcoming quarter.
7. The Program Manager signs in to the browser-based system by supplying their credentials to the website.

8. The Program Manager sees a message that notifies them that the system has been populated by the Program Administrator for a new quarter. He or she opens a list of all the courses in their department.

9. The Program Manager will select which courses will be offered for this particular quarter.

10. The Program Manager will also specify which instructor will teach each course.

11. After each Program Manager has selected each course to be offered and its corresponding instructor, the Program Administrator will be notified via e-mail.

12. The Program Administrator can then review the entire list of offered courses if he or she chooses. The Program Administrator will also be able to run the scheduler at this point. The scheduler will algorithmically determine when courses will be taught. The system will attempt to provide a best fit and present that to Program Administrator for review, modification, or acceptance.

4 Requirements: Specification

There are four sections to this system: the user interface, the database, the scheduler, and the connecting component.

The user interface must be very clean and user-friendly. It must be intuitive both to freshmen students of eighteen and to senior professors of seventy.

Program Administrators must be able to modify and perform operations on the database. Program Administrators must also be allowed to run the scheduler. They must also have a simple way to communicate with the Program Managers.

Program Managers must be able to retrieve course information specified by a Program Administrator. With this information, Program Managers

must be able to specify the courses for the upcoming school session. This selection must propagate automatically for the scheduler's usage.

Lecturers must be able to specify preferred teaching times and preferred courses. They must also be able to see which courses they have been selected to teach. This information must be automatically added to the database for the scheduler's usage.

Students must be able to see the scheduler's output once a Program Administrator has accepted it. They must be able to search, view, and select courses from the catalog. As students will traffic the system most heavily, the student's user interface must be especially simple and efficient.

As students, our team has an advantage in that we are keenly aware of the needs of this particular user role. We are capable of being both designing it from the vantage point of an engineer and testing it from the perspective of a student.

The database must perform read and write operations efficiently and reliably. It must be robust as potential user bases at some universities are quite large; load on the database will be proportionally large.

The scheduler must have an algorithm that is first able to handle multiple constraints. This algorithm must be scalable so that, given increasing constraints, it will remain both functional and efficient. If the algorithm cannot find an optimal schedule, it must be able to convey conflicts to Program Administrators and allow them to manually resolve these conflicts.

Lastly, there must be a connecting component that links the user interface, the database, and the scheduler. This connecting component, as with the rest of the system, must be robust under high loads, allowing for both larger and smaller systems.

1. User Interface
    (a) The web platform must be:
        i. Compatible with multiple browsers.
        ii. Intuitive and easy to understand.

(b) Views based on user roles:

i. Program Administrators must be able to:

A. Modify the database.

B. Choose constraints for the scheduler.

C. Run the scheduler.

ii. Program Managers must be able to:

A. Retrieve information provided by Program Administrator.

B. Specify the courses for a certain school session.

iii. Lecturers must be able to:

A. Submit preferences to the Program Manager.

iv. Students must be able to:

A. Enroll in courses.

(c) Must interact with the connecting component.

2. Database

(a) Must be able to store and retrieve data efficiently and reliably.

(b) Must handle load well.

(c) Must interact with the connecting component.

3. Scheduler

(a) Constraint-based algorithm:

i. Must be able to handle multiple constraints (e.g. room capacity, lecturer availability, subject preference, etc.).

ii. Must be precise in handling schedules for the Program Administrator.

iii. Must provide an output that is easy to modify, in case the algorithm does not provide a solution for a set of inputs.

(b) Must interact with the connecting component.

4. Connecting Component

(a) Must provide an interfacing between the user interface, the database and the scheduler:

(b) Must be robust to allow for systems both small and large.

5 Architectural Design

Our architecture is broken into four main categories:

1. User interface or UI (The front end)
2. The database (The back end)
3. The scheduler (The engine)
4. Django (The framework)

5.1 The User Interface

There are four different user interfaces. Each is assigned to one of the four roles that a user could be: Program Administrator, Program Manager, lecturer, and student.

The Program Administrator's user interface is fairly simple. They are given fields to edit elements such as courses, rooms, and professors. Once they fill out the fields for the master list, this list is propagated to the Program Managers. Once the Program Managers have selected course offerings for the quarter or semester, then the Program Administrator may run the scheduler. Once the Program Administrator has accepted the scheduler's output, the schedule will be propagated across the system.

The Program Manager's interface will display all of the courses and lecturers for their department. The Program Manager will have the option of selecting the lecturers and courses that they want to be offered during the quarter or semester.

The lecturer will have an interface that will hold the courses that they can teach and the current courses that they are teaching. They will be able to manually add or drop students from their courses, and assign grades to students in their courses.

The student will have a customizable interface which will hold their past,

present and future schedules. It will also have other widgets that they will be able to add to their interface.

5.2 The Database

Our database holds several different sets of tables. There is a "master list" set of tables that the Program Administrator modifies, and a set of tables that the scheduler outputs to. There are also tables to store individual user data, such as login credentials and course selections. To avoid redundancy, we use foreign keys often in our database.

The master list tables hold the following elements: School, Department, Class, Class lab, Prerequisites, Building, Room, Period, Lecturer, Person, Role, and Person Role.

The scheduler outputs to a set of tables that hold a complete schedule of the quarter or semester. Past schedules will also be stored in the database in the same format.

5.3 The Scheduler

The algorithm we use is a modified genetic algorithm written by Mladen Jankovich, a Serbian software developer. It mimics the process of natural selection in order to find a solution to our scheduling problem. The algorithm represents the variables as "chromosomes." A full set of chromosomes make up a parent. Each parent is given a fitness value according to how well they fit into the schedule. Fitness is higher for matches that fit into empty classrooms, have the appropriate number of seats, or if the classroom has a lab in it.

A group of parents make up a population. The algorithm takes 'n' number of parents from the population and does a crossover on pairs of the selected parents to create 'n' new chromosomes. The algorithm replaces 'n' chromosomes from the existing population with the new chromosomes that were created by the crossover. It does not replace the chromosomes with the best fitness. After the crossover is performed, mutations take place. A random

number is generated, which represents the mutation size. While the size has not been reached, classes are moved to different rooms randomly. Classes with the best fitness are ignored.

The algorithm repeats this process until it reaches a fitness of one hundred percent. This means it has found a solution that satisfies all of its constraints. The algorithm pulls information from the database, and enters it into four different lists: a professor list, a course list, a room list and a classes list. It then outputs into a second database which is then accessible from the user interface.

5.4 Django

Django connects the user interface, the database and the scheduler. Django manages the user interface by handling the user requests and providing the appropriate response.

For example, when the scheduler is called by the Program Administrator, the browser creates an HTTP request. Django receives this request and transforms it into an object. It checks the URL and determines to which function it must send the request object. The function in this case is the scheduling algorithm, which takes the information stored in the database and attempts to find a solution. The output of the scheduler is placed into a second database and a response to the request object is generated and sent to the Program Administrator.

6 Project Plan

Our project was part of a software methodology course and many of the deliverables submitted for the class are similar to those in the SCORE requirements.

Instruction began on September 23, 2010 and continued for ten weeks. Further development of our scheduling system will continue into the Winter quarter whose duration began January 4, 2011 and ends March 18, 2011.

Week 1: Initial Presentation

The first week we formed our group and gave an initial presentation on our proposed project. In the presentation we introduced our team's project name and group members, along with an overview of our project. The overview included our desired core functionality, our perceived project risks, our predicted user experience and our wish list.

Perceived Project Risks

Risk	Description	Risk Severity (1-5)	Resolution
Getting algorithm wrong or not finding a suitable one	Finding the right algorithm to run our scheduling engine might be difficult and time consuming.	5	We tried three different algorithms and felt the first two did not satisfy our needs and requirements, but the third did.
Learning Django	Django, the back end of choice, might be difficult to learn.	2	Our back end system developers took a reasonable amount of time learning to use Django.
Completing project before academic quarter ended	The scope of this particular project was large and there was a possibility that we would not complete it before the end of the quarter.	3	We were unable to finish the system. However, we did manage to develop the system in fragments and they worked to specifications. Development continued during break and into the following academic quarter.

Week 2: Requirements - Scenarios

Our team designed user scenarios related to our project. The scenarios were a significant part of our requirements document and were important in defining various aspects of our system's functionality. We came up with user scenarios for the following roles: Program Administrators, Program Managers, lecturers, and students.

Each scenario outlined and described the flow of data and the interaction with different functions and systems as the user performed a particular action.

Week 3: Requirements - Complete

After defining our scenarios, we created a complete requirements document which was necessary for implementing our system. It detailed the specification of the functionality of our scheduler and constraints on that functionality.

The requirements we described included the following:

- Functionality - A specific and detailed description and list of our system.
- Usability - How users would interact with our system.
- Coding Standards - Rules for organizing and formatting code.
- Preliminary User Interface - Preliminary mockups of what our user interface would look like.
- Wish List - System features we hoped to implement if time permitted.

We gave a design presentation which outlined everything in our requirements document and how we planned to do it.

Week 4: Architecture and Design Document

After defining our requirements, we created a design document that described the architecture of our system. It had a detailed description of the objects that we would use as well as relationships between objects.

We included the following items:

- Overview - The overview document provided context for our system and an overview of our diagrams. It included descriptions of our major design decisions.
- Architecture Diagram - High-level overview of the components in our system and how they co-operate.

- UML Structure Diagrams - Diagrams of our class objects that described the attributes and operations in Unified Modeling Language.
- UML Interaction Diagrams - A diagram that described several of the scenarios we produced for the requirements document.

Week 5-6: Development and User Manual

During this phase of our course, implementation of our system commenced and continued for the next several weeks. This included implementing our design and requirements document. As we created this system, we simultaneously created a user manual that detailed how our system would work. The user manual explained the user interface and provided a walkthrough for common tasks.

Week 7: Software Inspection

Our software inspection was an in-class presentation in which we systematically read source code. We identified and classified defects as we noticed them. For our software inspection, we chose to review our algorithm as it was one of the most difficult parts of our project to debug.

Week 8: Unit Test

Our group created several unit tests using Python, the Django testing suite, and a third-party tool called Selenium. These tested the various modules of our system both separately and in conjunction with one another.

Week 9-10: Acceptance Test

Our final deliverable was an acceptance test for our project. This deliverable was a working prototype designed to exhibit the same qualities that our

final product will have. It demonstrated most of our project's functionality, though it was far from feature-complete.

Because our system was still incomplete at this point, we decided to spend the next academic quarter completing development of our project. The next academic quarter at UC Santa Cruz began January 4, 2011 and would end on March 18, 2011.

The following is our plan for the second quarter.

Week 1: Review

Our plan was to review our system and deliverables. We defined items that remained incomplete or that required improvement. We also discussed switching from the Unified Process to an agile Scrum software development process. We revised our meeting schedule to include short check-ins every night along with a longer weekly planning meeting.

Week 2: Documentation Development

At the time of this report's submission, we are at this stage in our plan.

With our SCORE submission approaching, we focus this week on writing the necessary report. After this week concludes, we will continue development on our project.

Week 3-10: Continued Implementation

During this period, we will return to integrating the disparate modules of our system and adding features from our wish list.

7 Management Plan

Initially, we chose a democratic team structure. During our first quarter we would meet twice a week for a short period during our scheduled class time. We also met at least once each week outside of class. Meeting often was crucial in coordinating our project early in its development.

For the first quarter, course requirements guided our deliverables and project plan. It took a few weeks for us to settle into the team roles shown below. For the first few weeks of the project we were all involved in deciding how we were going to structure our system and what needed to be done. We would meet in class and discuss the research everyone had done since our last meeting. We then worked in pairs or individually until our next meeting. We waited until implementation phase to choose specific parts of the project that we would work on.

We revised our management plan at the beginning of the second quarter. We switched to the Scrum process - we determine as a group which features and tasks need to be completed, and then select them ourselves. We are now meeting once a day for fifteen minutes. We also have a weekly planning meeting. Will took over as project manager, and has taken on the responsibility of coordinating our meetings and resolving any issues that prevent team members from making progress.

The roles below are not exclusive; these are where our team members tend to focus.

Ben Ross - In charge of coding the algorithm

Erik Steggall - In charge of testing

Justin Lazaro - In charge of documentation

Sabba Petri - In charge of the user interface

Will Crawford - In charge of Django administration and overall expertise

Professor Charlie McDowell - Project reviewer

Professor Linda Werner - Project reviewer

7.1 Channels of Communication

Subversion (SVN)
As part of the class, we were required to use Subversion as a source control mechanism. It that allowed multiple people to remotely collaborate on the same codebase. We used Subversion to transfer code and miscellaneous files between team members. We also used it to automatically merge source code files.

Google Groups
For the purposes of communicating by e-mail, we created an e-mail list with Google Groups. This allowed us to bounce our e-mails to each team member. It also automatically archived our discussions. Meetings were scheduled primarily via this e-mail list. It also initially served as a rudimentary feature tracker.

Daily Skype Meetings
After the conclusion of the class, we continued to meet via Skype every night for 15 minutes to check in with each other. These meetings were intended to briefly answer three questions: "What did you work on today?" "What are you working on tomorrow?" and "Is there anything preventing you from making progress?"

Git & Codaset
After the class concluded, we switched from Subversion to Git as more of our team members were comfortable with it. Git is another source control mechanism, but we found its granularity regarding which files it committed to be superior to that of Subversion. We also found its merging algorithm produced fewer conflicts during our usage. The website Codaset (`http://codaset.com/`) served as a central repository for our highly mobile team, as well as a GUI for viewing the repository's metadata, such as commit histories, volume, etc.

8 Implementation

The overall system was broken down into three main parts: the web interface, the database and the scheduler. Our implementation strategy was to find the best way to implement these parts separately, and then collectively integrate them.

The front end of our system was written in HTML, CSS and Javascript. We picked HTML and CSS because they are standard for web design. We decided to use Javascript to add functionality to the front end because it is widely supported by browsers and renders much faster than Flash. In addition, unlike Flash, Javascript is rendered in browsers without a plugin.

The Django framework was selected for a few reasons. Django has an interface that allows for robust database operations without manually writing SQL queries. It also has a Pythonic template system to display information on dynamic web pages. The use of this framework allowed us to seamlessly integrate the scheduler with the database. We predicted that the integration of the scheduler with the database would pose a major risk to our system's development, but we managed to solve this early in the process. Django uses forms and templates to display information from the database to web pages. This allows us to create a few templates for each of the authentication levels.

The scheduler posed the greatest challenge in developing the system. We choose to implement the scheduler using a genetic algorithm. This algorithm was based off of a freeware genetic algorithm written in C++ by Mladen Jankovic.

9 Validation and Verification

Our test framework for our implementation has three components: testing for the scheduler, testing for the web interface and testing the database through the Django framework. The scheduler is tested by scripts, the web interface is tested through a software tool, and the database is tested through Django's testing suite.

The scheduler was first tested manually by checking the output for a valid schedule. This proved to be very tedious. The algorithm team developed an automated script that was able to check the schedule, verifying its validity. This script allowed us to verify the output from small datasets, but also scaled to verify large datasets as well.

The web interface testing was done using a framework called Selenium. This open source tool allowed automatic testing of the web UI. Two methods were used for writing tests for the web UI. Selenium provided an IDE that allowed us to record an interaction with the web UI and run it as a test. This was done for basic actions, such as logging in and out. The other way our Python scripts used Selenium's built-in libraries was to interact with the web interface. This allowed us to check for output on a given page. This was done for advanced actions, such as signing up for classes, or adding students to the database.

The database's testing is done through Django's test framework. The suite can test the framework with other utilities. The testing suite is split into two different sections: doctests and unit tests. The tests are run through Python's manager. Python's manager can test the entire database, or specific modules of it. Our project used the unit test portion of the test suite to test the database and its connection to the algorithm.

10 Outcomes and Lessons Learned

The development of this project has taught our team many valuable lessons in software engineering. First and foremost, we learned how to work together as a team to develop a software system. Previous to this project, none of the group members had developed a project of this magnitude. Learning how to divide tasks and break up the work based on skill sets of team members was something that we struggled with at the beginning of the project. Because we did not understand the magnitude of the project, we divided tasks based on member's interests. We soon realized this was not the best decision, and our initial development cycle was thrown off because we had to choose roles to which we were best suited, even if it was not our first choice.

Throughout the development of our system, the tools and technologies that our group was using were under constant change. Our scheduling solution changed three times until a suitable solution was found, as a result, there was a lot of time spent researching technology that was never implemented. Once we utilized our team member's skills to find the right technology, the development of our project became easier and accelerated. The combination of adopting a new development process, locking down the technologies needed, and understanding one another has increased our productivity and our understanding of working on a development team.