S.C.O.R.E 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 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 graduate student 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.

For our first deliverable, part of our inception phase, we wrote out scenarios for the use of our product. We included a scenario for each of the user categories that would be using our product. Our Program Administrator scenario described how the Program Administrator would initially set up the database in order to run the algorithm. The scenario for the Program Manager shows how the Program Manager should be able to inspect the courses that the administrator picked, and select the courses and lecturers for the upcoming quarter. The scenario for the lecturer shows how the lecturer should be able to specify their own individual constraints, such as the courses that they wish to teach, and the days and times that they are available. The scenario for the student shows how a student should be able to select a course for their upcoming quarter. We also included a scenario in which the Program Administrator can receive the master schedule from each individual Program Manager and is then able to set up and run the algorithm.

Our second deliverable provided an overview of our requirements, also part of our inception phase. In this deliverable we provided a broad overview of our system and its functionality. It was broken into functionality, performance, usability, constraints, and our wish list. The functionality provides a description of what our system does, we included information on the algorithm, a breakdown of our database, the different sections of our user interface, and the specifications of our server.

Our third deliverable was an overall summary of our project, part of our elaboration phase. We provided a high level architecture that showed the break down of our system into user interface, Django, and the algorithm. We went into detail on how we planned to build our project and reasons for building it the way we did. We also included interaction diagrams, in which we elaborated on the scenarios we selected.

Our fourth deliverable was a user manual for our project, part of our construction phase. This deliverable was primarily a tutorial for each user role showing how they will use our final product. It had a step-by-step process for setting up accounts and performing the necessary tasks for each role. Our user manual also includes a system overview so that it is more clear for the user reading it.

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.

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 → 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, it mimics the process of natural selection in order to find the 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 class 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, some 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, in which has it has found an optimum solution. The algorithm pulls information from the database, and enters it into four different list: a professor list, a course list, a room list and a classes list. It then outputs into a second database which is then displayed to 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. In the case of the Scheduler it takes the request from the Program Administrator to run the Scheduler and creates a request object and sends it through multiple middlewares (Python functions). Django then checks the URLs and determines which function it must send the request object. The function in this case is the Scheduler, which takes the information stored in the Program Administrators database and runs. The output of the Scheduler is placed into a second database and the request object is sent back up through the middleware in the reverse order of the way it came in. The second database is then displayed to the Program Administrator.

6 Project Plan

Our project followed a software methodology course and many of the deliverables are similar to those in the SCORE requirements.

Instruction and classes began on September 23, 2010 and continued for 10 weeks until the end of the quarter. Further development of our scheduling system will continue on to the winter quarter starting January 4, 2011 through March 18, 2011.

Week 1: Initial Presentation

For the first week, we had to gather in groups of 5 to form teams for our project and give an initial presentation on a brief overview of our project. In the presentation, we introduced our project's name and our team's members. We also introduced our ideas about the potential user experience, our desired core functionality, and the potential project risks we saw at the time. We also mentioned a few features that we thought would be nice to add if we had time for them.

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

During this week, our team designed several user scenarios related to our project. The scenarios will be a significant part of our requirement document and will be important in defining various aspects of our system's functionality. We came up with 5 user scenarios: (2) Program Administrators, Program Manager, lecturer professors, and students.

Each scenario outlined and described the end user's interaction with the system, the data flows of information, 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 will be necessary for implementing our system in a clear and concise manner. It details the specification of the functionality and constraints on that functionality for our scheduling system.

The requirements we cover include the following:

- Functional - A specific and detailed description and list of what our system is
- Performance - Specific characteristics of the performance of our system
- Usability - How users interact with our system
- Wish list - System features we hope to implement if time permits
- Coding Standards - Rules for organizing and formatting code
- Preliminary User interface - Preliminary sketch drawings of what our user interface will look like.

In this week, we also had to give a Design Presentation which outlined everything in our requirement documents and how it is done.

Week 4: Architecture and Design Document

After defining our requirements and scenarios, we created an architecture and design document that documents the high-level architecture and the detailed design of our project. It has a detailed description of the objects that we will use as well as relationships between objects.

We created the following items:

- Overview - The overview document will provide context and an overview of our diagrams. It will include descriptions of our major design decisions and modularization criteria.

- Architecture diagram - High-level overview of the components in our system and how they co-operate.
- UML Structure diagrams - Diagrams of our class objects in the form a UML structure diagrams that describe the attributes and operations.
- UML Interaction Diagrams - A developed UML diagram based on several of the scenarios we produced for the requirement documents.

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 requirement documents. As we were creating this system, we were also simultaneously required to create a user manual that details how our system is to work. The user manual was broken down into the following parts:

- Purpose - Description of the UI and functionality
- How do you do this? - Defining user classes, convey the user experience level at each of the classes, and provide computerized help.

Week 7: Software inspection

A software inspection is an in-class presentation where a group simultaneously and systematically reads source code and identifies and classifies defects as they are encountered. For our software inspection, we chose our algorithm class.

It is also important to note that a software inspection is strictly a time to catch defects and not make an attempt to discuss how to fix them or what to do to fix. There is no discussion of the problem at this time, it is simply to define the problem and make note of it to the developer.

Week 8: Unit Test

Our group created several unit tests using Phython, Django testing suite, and Selenium that tested the various classes and modules of our system. These simple scripts that automated and simplified testing for us.

Week 9-10: Acceptance Test

With our development, testing, and inspection complete, we demonstrated our system to our primary stake-holder (our professor). An acceptance test is generally a basic run-through of our entire end-to-end user experience of our system. At this point, the quarter is about to end and all our project deliverables have been completed and ready to be delivered as a final product. Due to the scale of our particular project, we unfortunately did not have a complete end-to-end system. However, bits and pieces of our system worked well when isolated.

Because our system was 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 tentative plan for the second quarter.

Week 1: Review

Our plan is to review our system and deliverables. We define items we need to complete or improve on. We also make the decision to revise our life cycle model from Unified Process to an agile Scrum software methodology. We plan on holding short meetings every evening to update each other with any new progress, along with weekly planning meetings where we sit down and discuss future development.

Week 2: Documentation improvement

With our SCORE submission coming near, we focus this week on writing the report. After this week has concluded, we'll continue development on our project.

7 Management Plan

7.1 An Overview of the Management Process

We chose a democratic team structure; the management is broken into two timeframes corresponding to our schools quarter system. For the first part of our project we would meet twice a week for a short period of time durring our scheduled class time; and at least once outside of class. Meeting often was crucial in coordinate our project early on in development.

For the first quarter, course requirements directed our deliverables and project plan. It took weeks for us to decide our roles (specified 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 best options from the research everyone had done before class and then work in pairs or individually until our next class meeting. It wasn't until the implementation phase that we each chose specific parts of the project that we would each work on.

We revised our management plan at the beginning of the second quarter: We switched over to the Scrum process, and we now are meeting once a day for fifteen minutes. Additionally, 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.

Ben Ross - In charge of coding the algorithm

Professor Charlie McDowell - Project reviewer

Erik Steggall - In charge of connecting Django and the scheduler

Justin Lazaro - In charge of documentation

Professor Linda Werner - Project reviewer

Sabba Petri - In charge of the user interface

Will Crawford - In charge of Django administration and overall expertise

7.2 Channels of Communication

Subversion (SVN)
As part of the class, we were required to use Subversion as a project repository for our code, as well as other deliverables for the class. Subversion is a source control mechanism that allows 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; it allowed us to automatically archive our e-mail conversations and bounce all communication to each team member. Scheduling meetings was done 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 and preferred 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 the 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 four main parts: Web Interface, Database, Scheduler, and Security. Our implementation strategy was to find the best way to implement these parts separately, and then collectively integrate them together.

The front end of our system was written in HTML and CSS. We picked HTML and CSS because they are the standard web design languages. We decided to use Javascript to add functionality to the front end because it is widely supported 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 as well as a Pythonic template system to display information on dynamic web pages. The use of this framework us to seemlessly integrate the Scheduler with the Database. We identified the integration of the Scheduler to the Database to be a major risk facing our system development, but was mitigated early in the process. Django uses forms and templates to display information from the database to web pages and this allows us to create a few templates for each of the authentication levels, all of which are able to display information directly from the database with no SQL queries. Because Django performs operations on the database transparently, Python queries are performed with great speed, letting pages load quicker.

The Scheduler was the biggest risk of the development of the system. We choose to implement the Scheduler using a genetic algorithm. This Algorithm was based off of a freeware genetic algorithm scheduler[citation goes here]. The Algorithm was written in Python, allowing it to be fast and easily modifiable. The genetic algorithm was selected as the best implementation solution, because it allowed seamless integration with the database, and it proves incredibly fast, even for larger solutions.

Security is one of the most important aspects of a web application, especially one for universities.Our first important security feature is HTTPS. HTTPS is used because it allows all traffic to be encrypted. The encryption capability is a step above popular web applications like Facebook, and Myspace. Our second important security feature is passwords. Our minimum password length is 8, with a minimum of one capital letter, one symbol, and one letter. This gives users increased security from brute force password attacks. Another security feature involving passwords is similar to what some banks do. We assign each user a sitekey, which is a random picture out of 50. The user is then requested to name the sitekey. When the user enters their password, if an incorrect sitekey is displayed, the user is made aware of an attempt to compromise their information.

The fourth other security features deal with the privacy of the user once they are in the system. These features are designed for the student role only. The integration of Facebook with our system provided another security hole. When the student first logs into the system, a tutorial walks them through setting up their account and linking it with their Facebook account. The student has the option to turn off sharing of information during this setup, as well as, anytime through the account settings page. This feature gives permission to students for complete access as to what information they share through our system, and through Facebook. Allowing students this control will decrease the chances of a student sharing more information than they intend.

9 Validation and Verification

Our test framework for our implementation has three forms. Testing for the Scheduler, testing for the Web Interface and testing the Database through the Django framework. The testing for the Scheduler is done by scripts,the Web Interface testing is done through a software suite, and testing the Database is done 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, and long. The Algorithm team developed an automated script that was able to check the schedule, verifying it was valid. The automated script allowed testing of much larger datasets, which helped prove the validity of the Scheduler.

The Web Interface testing is done using a framework called Selenium. This open source tool allows automatic testing of the web UI. Two methods were used for writing tests for the web UI. Selenium has an IDE that allows you to record an interaction with the web UI and run that recording as a test. This was done for basic actions, such as logging in and out. The other way our python scripts that use Selenium's built in libraries 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 and 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 the database to algorithm connection.

10 Outcomes and Lessons Learned

The development of this project has taught our team many valuable lessons in software development. First, and foremost is learning 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 onset of the project. Because we did not understand the magnitude of the project, we divided tasks based on member's interests of specific tasks. We soon realized this was not the best decision, and our initial development cycle was thrown off because we had to choose roles that we were best suited, even if it was not our first choice.

After using the initial development process for ten weeks, our group decided to change our development process from the Unified Process to Scrum. This decision was made, because our initial process was not suited for the fast paced development that our system needs. The Unified Process had slow increments,the long time delays between due dates allowed members of our group to fall behind on the project. Our new method allows us to keep all members accountable, and to keep everyone updated on the current status of the system. Realizing that one development process does not fit all was a great lesson learned through this project.

Throughout the development of our system, the tools and technologies, that our group was using, were under constant change. Our solution to scheduling changed three times until a suitable solution was found, as a result, there was a lot of time wasted researching technology that was never implemented. Once we were able to utilize our teammember's skills to find the right technology, the development of our project became easier, and was quickly gained speed. 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.