OnTarget

A Course Scheduling Tool for College Students and Advisors

Final Project Report

Belinda J. Copus

Team # 24

CS 5551, Advanced Software Engineering

Spring, 2017

Project Description

OnTarget is a software application that automates the development of a plan of study for an undergraduate student. This application uses real-world actual data to create a realistic plan according to the constraints provided by the student. This application will run in a web browser and be useable on the desktop and mobile devices.

Introduction

College students face many challenges transitioning to university life, including planning what courses to take each semester. Students are often unaware of the courses that need to be taken, what prerequisites are needed, and when these courses will be offered. It has been stated that academic advising is a vital link in retention (Drake, 2011). A logical conclusion could be made in that a student without a well- defined plan of study is less likely to complete a degree. Long range course offering schedules often are not available for students to access and create long range plans of study, independently, nor with any confidence that the courses they choose will be offered in the semester they believe it should be taken. Also, it is increasingly likely that a university will offer the same course on more than one campus and students often wish to take classes on more than one campus. Students must rely on a faculty or academic advisor to assist them in planning their coursework, but few students actually regularly meet with their advisor. Advisors also find the task of drafting a plan of study for the student to be quite tedious and time consuming. For an advisor to create an optimal plan that accounts for the many variables involved, is nearly impossible. Often, students have unique situations and constraints, adding even more complexity to the task. A software application that automates this effort is needed by both students and advisors.

Project Goal and Objectives

OnTarget is an application that will facilitate the task of creating a student's college course plan. There is an additional side benefits to the departmental administrators, in that, a sense of future course demand can be determined by this application.

There has been great effort made in creating automated advising tools, similar to this proposed system. There have been systems which are prescriptive, in that the student is provided a plan of study with little input. Others have created web-based decision support tools for academic advising is based on promoting engagement between the advisor and student (Feghali, Zbib, & Hallal, 2011). The system proposed is not necessarily unique in capabilities, but will be offered free and openly for those who would like to use the application. This system is superior to many readily available systems, in that it is able to consider distant future course offerings, whereas most currently marketed systems only consider a scheduling from a one-semester look ahead.

This system is vital to saving time and providing a more accurate plan of study than what a person is able to create. It is the objective to develop and deliver this system for use by Fall 2017. While this proposed system is intended to be used by Computer Science students and faculty at a small university, it is possible that the application could be extended and adapted to other schools and degree programs.

System Features

The system will utilize the following data:

- Course catalog major requirements and general education requirements,

- Student's completed coursework and major/concentration,
- Course ranking per course in terms of time requirement outside of class,
- Course offering 5-year rolling plan, and Prerequisite mapping.

This list comprises the foundational data used by the system.

A student will be able to generate a complete plan of study that leads to graduation. The student will be able to indicate which campus they prefer to take courses, the number of hours they wish to undertake during a semester, or whether they would like to take classes during the summer. General education requirements will be scheduled in addition to the requirements specified by the major. The student will also be able to specify load balance, in that the student can choose to balance course load between demanding courses and less demanding course. The plan will be stored so that it can be accessed by the student and/or advisor at some later time and by different hardware platforms.

The system will notify the advisor and student if a change is made in the 5-year rolling plan or in a pre-requisite requirement that will impact the current plan. Additionally, if the student is unable to obtain a sufficient grade in a course, the student and advisor will be notified and the plan can be automatically revised. If a student does not follow a plan for a particular semester, the system will automatically adjust the plan to reflect the shortfall.

The system will also track the number of students expected to take a particular course during a given semester to assist in allocating resources.

This system will web-based and be able to be utilized on multiple platforms, i.e. desktop, Android, and iOS platforms.

Project Management Report

Target functionality and date for delivery:

The goal for deliverables was a functional system ready for use beginning in August 2017. The following use cases were to be completed:

- Administrative – course plan 5-year plan for offerings loaded, programs for Computer Science (5 tracks) and Cybersecurity (2 tracks), prerequisite requirements, and general education requirements. Initial student data (course completion), registration and login functionality
- Advisor – view/print/save a plan for a student, modify a plan for a student, update student course completion, generate a plan base on constraints
- Student – view/print/save plan, change preferences, generate plan of study

Please see use case descriptions in the Appendix, Third Increment Report.

Diary of Progress

There were four increments for this project that were spread over a three-month period.

Increment	Objective	Number days, including weekends
Increment 1	Project plan and initial development	18
Increment 2	Development	20
Increment 3	Development	30

Increment 4	Project clean-up and deployment	15

Summary of activities completed in each increment:

Increment 1

- Use cases defined, initial schema for database created. Established Project management using Zenhub.

Increment 2

- Algorithm design for creating a plan of study from the data inputs. Realized brute force was not a good option.
- Explored implementation of the database using MongoDB.
- Explored and began implementation of application using Feathers.
- Discovered Mondo DB was not the right choice and switched to MySQL. Also during this increment, I was introduced to Express and Ionic and it was determined that this would be a better choice to get use cases produced faster.

Increment 3

- Explored using Particle Swarm to generate plans of study and arrive at an optimal plan.
- Worked on user interface for the advisor and student views.
- End-to-end connection, round-tripping, for the following:

Increment 4

- All about getting discrete particle swarm to work for the application.

Technology

During Increment 2 it was determined that a brute-force method of generating an optimal plan of study was not practical. This enhanced understanding of the problem significantly expanded the scope and size of the development effort for this project.

Discrete scheduling problems are notoriously difficult: Given a set of elements that must be scheduled, and a set of constraints, determining an order that satisfies all constraints and “visits” all the items is essentially a constrained Travelling Salesman Problem, which is discussed extensively in the computer science literature. The difficulty of such a problem is O(n!), where n is the number of elements to be scheduled.

The essential problem at the core of the OnTarget functionality, as envisioned, is to schedule a large number of courses across a limited number of semesters, with various hard constraints. Some of the constraints include course prerequisites, semesters a student does not intend to take courses, campus locations, min/max/preferred hours of study, semesters a course is planned to be offered, constraints by course catalog year, and specific constraints for each concentration within a degree area—this for seven programs, under four catalogs, and about 125 individual courses.

The problem is additionally complicated by the fact that many degree requirements can be met in a variety of ways. All course that are possible to take are not obviously required for a degree; some are required, some are optional, some may fulfill (or partially fulfill) various requirements of the program of study. It is the very complexity of this problem that resulted in the desire for a tool like OnTarget, but also has complicated the implementation of OnTarget.

The scale of a brute force attack on the OnTarget scheduling problem is O(n!), where n is the number of courses that can be scheduled in a semester, times the number of semesters of a course of study—in other words, perhaps O(50!) at a minimum for a four-year plan of study. Because a brute-force scheduling algorithm is impractical for a schedule of the length and complexity of a degree plan, it was determined that an optimization algorithm suited to high-dimension discrete combinatorial problems would have to be employed. Discussion with a colleague suggested that a discrete Particle Swarm Optimization system might fill the need.

Overview of Classical Particle Swarm Optimization (PSO)

The classical particle swarm system is designed for optimization of high-dimension continuous functions. Although the scheduling problem is discrete, not continuous, a review of the continuous algorithm is helpful before discussing the algorithmic modifications necessary to operate on discrete problems.

Particle Swarm Optimization is very straightforward in concept: A search space is envisioned that represents the possible values for all of the variables of the function to be optimized. A population of “particles,” typically 20 – 40, is instantiated so that the particles occupy positions in the search space, and have velocities by which their positions change in the search space. The position of each particle is represented by a vector of real values representing each of the *d* variables in the system, and likewise, the velocity of each particle is a *d*-valued vector. The search space can have as many dimensions as there are variables that are thought to be related to the solution, and each dimension will have a region of interest, i.e., a region within which the solution may be expected to exist.

In addition, each particle has two *d*-valued vectors that represent the particle’s memory: one that contains the position in the space for which that particle observed the lowest value for the cost function (“personal best,” p_{best}), and another that contains the “best” position in the search space of which the particle is aware (“neighborhood best,” n_{best}). Each particle informs some number of other particles about its personal best position at each iteration, and is informed by some number. The graph of these “informer” relationships creates “neighborhoods” of particles, but the informer relationships are created in such a way that they are not exactly bilateral, so that the global best information eventually, through the course of some number of iterations, makes it to every particle.

Initially, the positions and velocities of the particles are set at random. The system will then iterate some number of times until either a maximum number of iterations have passed, or a solution has been found to some degree of assurance. At each iteration, each particle will update its position, calculate the cost of its current position, update its velocity, and communicate its p_{best} with particles it informs. In pseudocode:

```
initializePopulation;
repeat…
    for each particle:
        update position;
        calculate cost of this position;
        update velocity;
        inform neighbors;
        update memories based on information from neighbors;
…until termination condition is met.
```

Update Position and Velocity
Position $\vec{x}$ and velocity $\vec{v}$ are updated each iteration as follows:

$$x_i \leftarrow x_i + v_i$$

$$v_i \leftarrow c_1 v_i + c_2 \cdot r(0,1) \cdot p_{best_i} + c_2 \cdot r(0,1) \cdot n_{best_i}$$

Note that the new velocity is a combination of inertia, the remembered personal best position, and the remembered neighborhood best position. The constants c_1 and c_2 are "confidence" parameters that express the maximum weights given to the inertia of the particle and its two memories. In practice, these parameters are related to one another in a complex way and have themselves been an optimization problem that has seen a great deal of attention in the literature. Typical values are 0.7 and 1.43 for c_1 and c_2, respectively. The function *r(a,b)* returns a random number between *a* and *b*. It is this random element that generates significant additional complexity in the search strategy; the randomness has been demonstrated to be essential to the algorithm's success in hard optimization problems, as it encourages nonobvious exploration of the search space.

Calculate Cost
To calculate the cost of a position there must be a function defined for that purpose. Developing the cost function for a real-world optimization problem is the most essential step, and often represents the largest part of the work. The cost function is calculated for each particle on each iteration.

Inform Neighbors/Be Informed
Each particle informs some number of neighbors about that particle's p_{best}. Each particle, when informed about another particle's p_{best}, will remember it as n_{best} if it is better (lower cost) than what had been remembered as n_{best} before.

PSO for Combinatorial Problems

The problem at the heart of the OnTarget scheduler is a discrete, combinatorial problem—that is, the variables can take only certain, discrete values.

The OnTarget scheduling problem is modeled as a set of "course slots", with several per semester, that can either be "null" or else contain a course, such as "CS 1110, Programming II, 3 credit hours" (in actual practice, the "course" could also be more of a "metacourse," such as "GenEd Competency IV elective, 3 credit hours"). The objective of optimization is to develop a plan of study that minimizes time-to-graduation under the catalog, program, and concentration, in light of a series of hard and soft constraints, and given a set of courses already completed by the student.

In classical PSO, concepts like "position" and "velocity" and "particle" are just metaphors, of course; as we move to combinatorial problems, those metaphors no longer serve. In particular, the idea of a "velocity" breaks down in discrete problems. The solution is to redefine velocity so that, instead of representing a rate of change in various dimensions, it now represents a series of juxtapositions of elements within the discrete space.

In the terms of the OnTarget scheduling problem, a velocity element represents swapping the positions of two courses in the schedule, and a velocity "vector" is just an ordered list of such juxtapositions. This modifies the behavior of the algorithm so that the information shared between particles is about whether schedules that have course A before course B are generally better (in terms of optimizing the cost function) than schedules with course B before course A.

An enormous amount of time has been spent during Project Iteration 4 in developing the velocity updating function. Many variations were tried. The initial version followed the sketch of an algorithm given by PSO luminary, Maurice Clerc, in his definitive book on the subject. It turned out that his sketch of an algorithm did not result in convergence of the swarm. In consultation with a colleague, the author then developed a new velocity updating algorithm that resulted in rapid convergence on the symmetric Travelling Salesman Problem. The traditional components of velocity remain as in classical PSO, but are carried over into dPSO as a probability that each component from the three velocity “vectors” will be included in the new velocity vector at each iteration. The traditional PSO randomness is of course included to encourage exploration “off the beaten path” of the search space.

OnTarget dPSO Implementation

The discrete PSO (dPSO) implementation for OnTarget was implemented in Java in an effort to reduce project schedule risk, since the author is most familiar with that language. The intent is to deploy the PSO system on the server within a web container (Apache Tomcat); the scheduling services will be accessed by the OnTarget controller through a simple REST API. Both the controller and the PSO system have access to the MySQL database through the appropriate connectors.

One of the objectives of the dPSO system was to implement the discrete particle system in a generic manner to reduce coupling with the OnTarget system, resulting in the potential for reuse. This also will allow easy integration of other trial optimization algorithms. To this end, an interface, `DiscreteOptimizableProblem`, was defined as follows:

```
public interface DiscreteOptimizableProblem {
  double objectiveFunction(int[] position);   // the function to be minimized by optimization
  boolean isFeasiblePosition(int[] position); //returns true iff the position does not violate any constraint
  Tuple[] getRanges(); //returns tuples indicating min and max integral values for each dimension
}
```

Any discrete optimization problem that can implement the three required methods, and can model its discrete space as an array of integers, can be optimized by the algorithm. dPSO is also general enough solve problems where the various dimensions have different discrete ranges: the `getRanges()` method returns an array of `Tuple` of `(int, int)` indicating the minimum and maximum values for each dimension.

The dPSO system was first tested on larger instances of the symmetric Traveling Salesman Problem with known optimal solutions, and it performed very well.

The OnTarget dPSO system currently represents a few thousand lines of Java code, and is in a functioning but incomplete state. It is generating student schedules, but they are not yet very good. The deficiencies are believed to be related to the modeling of the problem space and the modeling of the cost function, and are under investigation.

Architecture

OnTarget is implemented as a web-delivered, client-server, model-view-controller architecture.

Model-View-Controller (MVC)

To separate application concerns and reduce unnecessary coupling between components, OnTarget follows the MVC architectural pattern.

The Client applications are intended to be as thin as practical so that application logic is concentrated in the Server component. This will make deployment and maintenance easier, since bug fixes and feature updates can often occur without client-side changes. The Ionic Framework was chosen to implement the client-side functionality because of its mobile-first philosophy and ability to produce native mobile applications with platform-specific look-and-feel across the most popular mobile operating systems (Android, iOS, Windows Mobile). Ionic can also be deployed in the web browser for use on the desktop.

There are three client applications intended for OnTarget: Student, Administrator, and Advisor. The Student client is of course for students to use, and has rather limited capabilities as discussed above. The intention for the student client is to deliver it via dedicated mobile applications and via the web browser with a consistent interface. The Administrator client has not yet been implemented. It will eventually allow administrative functions on the database. The Advisor client will expose the core functionality of OnTarget to create and manage student plans of study.

The Controller functionality is provided by a server application built on the Express Framework, running under NodeJS. Express makes it very easy to expose a REST API and to integrate with a data store. I had initially found a framework called Feathers that is built upon Express and socket.io, and the first Controller instances used that framework. As I became more familiar with Feathers and Express, I realized that the additional functionality provided by Feathers was superfluous for this application, and decided to migrate down one layer of abstraction to Express.

The discrete particle swarm optimizer (dPSO) is implemented in Java as mentioned above. It has not yet been connected to the Controller, as it is still in early development. The intention is to deploy the dPSO system in an Apache Tomcat container and expose a REST API that is accessible only to the Express server component. When queried by the Express controller, the dPSO component will pull required data from the database, produce alternative plans of study by swarm optimization, and store the work product in the database. Communications between the two components will thus be kept small.

The database is implemented using MySQL on the same server as the Express controller and the dPSO system. I had considered MongDB for this application, and after spending some time on it, determined it is better suited to less structured content than this application. The MySQL database for OnTarget currently consists of 15 tables. Around 1000 carefully-constructed lines of SQL were produced to provide initial datafill for testing purposes.

The design philosophy that I am pursuing for queries is that as much logic should be pushed to the database as is practical, to reduce the footprint of the controller. I’ve learned a lot of SQL this semester (and spent some time on the learning curve) as I’ve built increasingly sophisticated queries, whereas previously I might have used multiple disjointed queries, and processed the result sets in the database client code (Java or JavaScript).

One area of significant development that is ongoing is the object-relational mapping from SQL to Java (for the dPSO system) and SQL to JavaScript object (for the controller).

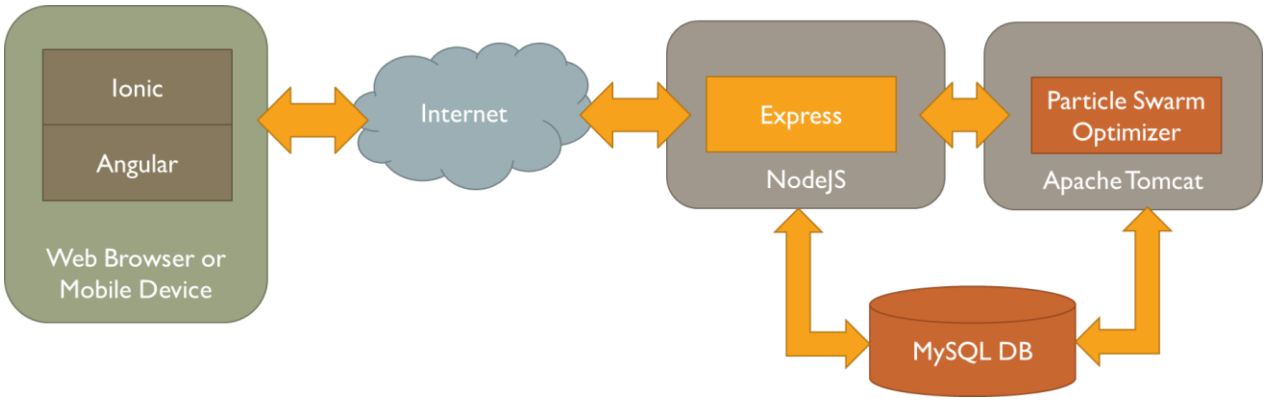


FIGURE 1. ONTARGET MVC ARCHITECTURE

Reflections on the Architectural Choices

I am generally very happy with the architectural choices made in this project thus far: Now that I have climbed various learning curves for the frameworks and technologies, I have existing patterns that I believe can scale. Express has been particularly straightforward to learn and implement.

The only area where I anticipate changes to the architecture are on the client side. The mobile-first design paradigm provided by the Ionic framework is going to be ideal for the Student client application. However, it is a bit constrained for the Advisor and Administrator clients. My plan is to spend additional time on Angular this summer, and pursue development of the more full-featured clients at that level, without the mobile-first look-and-feel of Ionic.

Final Project Evaluation

Applying the Agile Software Development Process to a medium-sized project with a team size of one

The Agile Software Development Process was initially applied to this problem. In the end, it was determined that the Agile method was not well suited to this problem and this team for a variety of reasons.

The primary hindrance was a lack of knowledge in terms of skills and algorithms to solve this problem. Initially I thought there was plenty of time to develop each of the use cases, but I greatly underestimated the time needed for the learning curve. The Agile Process thrives on being able to show progress, with the rapid completion of stories being the chief metric. I initially decided to produce the visual artifacts, i.e. user interface and data displayed to the screen. I quickly determined that this was creating substantial technical debt and was wasting a lot of time, since—like most real-word, non-synthetic problems—the user interface is just the proverbial tip of the iceberg. The real work for OnTarget was always going to be on the server side.

Elbanna and Sarker (2016) state that in Agile Software Development, "[technical] debt can quickly accumulate owing to the need to significantly reduce development time" (p. 74). I definitely experienced this issue first hand in my project. As a side note, teams seeing progress are more energized to tackle the next set of stories, or "progressive small chunks," as Coleman describes each iteration. "Satisfying the customer is very rewarding for the agile team" (Coleman, 2016).

For this reason, I have observed that Agile is a great choice for projects that are primarily focused on using existing APIs, creating stellar user interfaces, and are developed in an iterative nature—that describes most of the projects of my classmates. However, this wound up being not so transferrable to my project, which is much more back-end driven, and thus not as easy to develop in small iterations of progressive complexity. Rather than jumping in and showing visible progress, such as instant user interface, a great deal of time had to be dedicated to designing how data would be stored and accessed. This resulted in a huge technical debt for this team of one and resulted in an immediate halt in progress.

Reflecting at the end of this project the following thoughts come to mind: Implementing the Agile process for this project was an unmitigated failure because I was unable to produce a fully working product that delivered business value at each iteration, which is a key tenet of Agile development. I began to consider whether others had encountered a similar experience or could provide some insight to my perceived failure to produce a fully functioning artifact. This reflection has resulted in some additional study on the question of what can go wrong with Agile methodology. As a framework for my further reflection on this, I include some insights from Coleman on the four statements that form the Agile Manifesto and some brief reflections on my own experience through this project.

Individuals and Interactions over processes and tools

As Coleman states, “In Agile, the focus is on the team members and their abilities” (Coleman, 2016), emphasizing that large tools, comprehensive documentation and intricate, formalized procedures should give way to competent teams making measurable progress on short time scales.

For myself—the smallest conceivable team—interactions were essentially eliminated; the low-tech list on paper was enough to keep priorities in order. During implementation, the list changed with items being reordered. Sometimes the reordering was made to maintain progress, for example when I got stuck with the implementation of the particle swarm algorithm, I worked on user interface, and then returned after brief break. While this value statement is best understood in the context of a multi-member team, I found it does minimally apply even to a team of one.

Working software over comprehensive documentation

“Agile prides itself on regular delivery of the desired features to the customer” (Coleman, 2016). A working product produced early and often is prized over an overly large amount of time spent in documenting the development artifacts.

For this project, regular delivery of desired features was challenging. Initially, a demonstration of the user interface was made, but it was very artificial in nature. The client view needed to display real data! To do that, the client really needed to connect to a working server-side controller, which in turn needed to be wired to a functioning database. The database needed to have real data to access, which required the design of an appropriate schema, and also the production of a lot of datafill for testing. The datafill production was painstakingly done, since there are currently fifteen tables in the database, and a plethora of foreign keys with model-driven constraints. To add in the functionality to access the database was time consuming and results were slow at best.

Additionally, to create and display a plan of study, logic had to be created to build a plan of study. There was a tremendously long development cycle to design and implement the particle swarm algorithms for a discrete problem as described above—this is really breaking new ground, and constituted experimental development. In the midst of these things, what constitutes a rapid iteration that can be produced by a solo team that is fully-functioning and delivers business value? There are applications where the minimum viable product is larger than any artifact that can be produced in a single Agile sprint—this represents a mismatch between the product and the methodology.

The problem of delivering intermediate, functioning artifacts was further complicated by the learning curve for the various frameworks, i.e. Angular, Ionic, Feathers, Express. The burden of learning a framework and designing an algorithm to efficiently solve this problem caused significant delay in the production of working artifacts so highly prized by the Agile methodology. The burden was further increased by pursuing various frameworks and data stores (in my case, Feathers and MongoDB) to later learn that it was not the best choice for this problem. It is definitely to the advantage of an Agile team to have members already in possession of skills and tool experience, otherwise the frequent production of working software is greatly hindered.

Customer collaboration over contract negotiation

In Agile the customer (the "Product Owner") becomes one of the team. The customer is frequently in communication with the team collaborating on the next step, refining requirements, and providing feedback on software as it is produced. This relationship between the software team and customer enables Agile to be highly effective and avoid much late-term rework or failure to meet customer expectations. In my case, the customer and developer were the same individual. It could be considered that the instructor was also a quasi-customer, since delivery dates were set by the instructor. If this project had been produced in real-world conditions the customer and team would have worked to adjust delivery dates when significant obstacles were encountered.

Responding to change over following a plan

Agile's greatest strength is the ability to respond to modifications in customer requirements. In my project, little was added to the initial requirements. As development progressed, new features did come to mind, with the goal of adding these features in a future revision of the product. Had the project been extended in terms of time, these features could have been considered. Please see the final section for future features for this project.

Conclusion: Agile for This Team and Others

The single biggest challenge in this project was a lack of knowledge and experience, which led to failures to accurately estimate effort. This resulted in overcommitting the team in each sprint.

Had there been more prior knowledge of tools, languages, architectures, frameworks; and algorithms for scheduling problems, there would have been more opportunity for this project to be completed in a purely agile way, with appropriately-sized sprints and better intermediate artifacts. Instead, production of artifacts was delayed and halted by the need to learn. As a result, I failed the task of producing a working product in the allotted time. Nonetheless, this project was far from a failure. I have learned a great deal about both the weaknesses and strengths of Agile.

Since I was working alone on this project, the only reasonable response to knowledge deficit was to study and gain the needed knowledge before continuing. It occurs to me that this experience is not limited to my endeavor, or only to solo teams. In large Agile endeavors, how are knowledge deficits addressed?

The literature has much to say about the topic of Knowledge Management. One article suggests several ways for inter-team Knowledge Sharing (Santos, Goldman, & Cleidson, 2015). For example, it is suggested that there be a list, repository, or directory, be maintained of the skills and problem experience of each engineer. This list would enable teams to locate other engineers within their organization that could provide missing expertise is a given area or access others solutions to the same issue. This is a very simple way to address this issue and could be easily implemented in any organization. Another suggestion concerns the adoption of practices for socializing knowledge. The practices include removing barriers to, and encouraging, face-to-face communication. The article suggests having a physical workspace without walls or the utilization of glass walls between cubicles so that there is a visual connection between team members. It also suggests integration of areas that would draw team members (for example, with food), creating a space where conversations could be held and organizational members could learn about each other and their knowledge areas (Santos, Goldman, & Cleidson, 2015, p.1020). These ideas rely on the social nature of individuals and suggest intentionality in establishing trust within and across teams so that there is an integral, organizational perception of individual knowledge areas. "Knowledge sharing is one of the most important knowledge management processes, which gradually improves an organization's production system and its elements" (Santos, Goldman, & Cleidson, 2015, p.1007).

While these concepts seek to capture and understand the unknown, but already possessed, skill sets within an organization, there is also the inevitable case where no one in the organization has the knowledge needed to implement the solution. How does an agile team make essential and committed progress with a knowledge deficit?

One piece of research states that, instead of expecting to have a perfect team in possession of all necessary skills and abilities, "we need learning teams – teams that can repeatedly bend and blend on demand to suit the environments they work in (Rashina, Babb, & Norbjerg, 2013, p. 95). This group of authors state that the "biggest obstacle to such learning is iteration pressure – [the pressure to produce results each iteration] …[which] leads to sacrificing core agile principles with regard to continuous learning" (Rashina, Babb, & Norbjerg, 2013, p. 95). The authors also put their finger on another issue that is likely to forcibly collide with agile values: customers are paying for a product and are reluctant to fund time for learning (Rashina, Babb, & Norbjerg, 2013, p. 96). This tension between iteration pressure and continuous learning is thus very tangible. The article suggests a few ways to balance this tension. Clearly, managing expectations for the customer for the initial increments is a good, first approach. They suggest educating the customer and setting expectations that the initial sprint will be blown and not at all impressive. While the customer is on notice that the first increment will be minimal in terms of deliverables, the team has been given a little time to fill the knowledge deficit. (Rashina, Babb, & Norbjerg, 2013, p. 97). While this suggestion (and others in the article) are perhaps helpful, one is certainly left with the impression that knowledge deficits will frequently arise in agile projects, and will often result in culture clashes between the agile team and its customers.

After the knowledge deficit, the second obstacle for OnTarget was the time constraint, coupled with the very large scope of the problem to be solved. As my project arose from an actual application need, rather than just a desire to complete a requisite assignment, it was not chosen with much consideration for the time that would be allowed for implementation. Kovitz's study (2002) describes my situation exactly,

> "Often programmers work on tasks they've never encountered before. The majority of their time is spent thinking and understanding, not typing in source code. Once they've found a description that fits the task well, they can code it quickly. Before they've found such a description, predicting the time required for the job is difficult". (p. 141)

Much of the development time on this project was spent in figuring out how to tackle the problem. It was impossible to determine how long this learning period would be and that the learning period would exist at all.

My conclusion is that Agile Software Development is only truly effective when knowledge is acquired and architecture is at least somewhat established prior to implementation. During the course of this semester, I often wondered if there was a hybrid development model: A model that takes the best from traditional models, i.e. waterfall and the best from Agile. Perhaps in such a model the system architecture would be established through a more traditional software process, and implementation would follow in the Agile manner. With such a methodology, the foundation would be more firmly laid so that implementation will carry less schedule risk and less likelihood of large knowledge deficit or significant technical debt. Perhaps the initial design phase, "Increment 0," would produce design artifacts only, and should be allowed more time than a typical agile sprint; and the further increments would be 1-3 week agile sprints.

Bose and Thakur (2013) have proposed an interesting, hybrid approach along those lines. Their intention is likewise to take the very best from Agile and Waterfall methods and suppresses the weaknesses in both methods. In their methodology, documentation is performed in greater detail than recommended by agile. A Picture Plan is created in which, "infrastructure architecture decisions are made to address how the system will meet the defined functional, physical, interface, information protection and data requirements" (Bose & Thakur, 2013, p. 309). Rather than deciding on these details as they would arise in an agile project, they suggest that this step, which normally occurs in the Waterfall method, be given top priority and then be an input to the implementation stage of a project. Their methodology has other interesting features. For example, they would allow the client to make changes to requirements at any point during the project only if the change could be implemented in the time allowed by the next sprint. Hybrid software process models like this one may help to mitigate the issues of knowledge deficits and larger minimum viable products much better than social tricks, such as what was proposed by Rashina, *et. al.*

Final thoughts

I had previously stated that I thought this project was a failure since a fully functioning artifact was not produced. But upon reflection, I realize that I have learned much about Agile, in terms of it weaknesses and strengths that reading literature could not practically provide. My project will continue with a goal for an December, 2017 delivery. Now that the difficult issues have been resolved and knowledge deficit filled, implementation can proceed in a true agile-like manner. CS 5551 has been a great experience that all novice and more advanced software developers should experience.

Future direction of application

OnTarget is an ongoing project; I still need this application to facilitate my student advising activities. On the immediate horizon are the following priorities:

- Fully define the APIs for the following interfaces:
  - Student client to Express Server
  - Advisor client to Express Server
  - Administrator client to Express Server
  - Express Server to dPSO Server
- Begin re-implementation of the Advisor Client using Angular (instead of Ionic's mobile-first design).
- Complete work on the object-relational mapping for the controller and dPSO.
- Continue work on the particle swarm system (dPSO) to enhance its performance in generating near-optimal plans of study.

Some additional features beyond the initial use cases have also been identified. Among them are these:

- Enable a hypothetical plan to be created for a student considering transferring to our program. Often a transfer, or degree completer, will need to know how many credits per semester and how many semesters are needed to complete the program, before they apply to the University. These students have non-traditional lives and history, and need the information this plan will provide to make choices that will impact their finances and time.
- Learn about a student's abilities and adjust the plan going forward. This feature would use machine learning to classify students based on survey questions and past performance, and then adjust the plan of study to play to a student's strengths (and mitigate their weaknesses).

Links

Github:	https://github.com/bcopus/CS5551Project
Youtube:	https://www.youtube.com/watch?v=OXThNF8cvTw

References

Bose, L. & Thakur, S. (2013). Introducing agile into a non-agile project: Analysis of agile methodology with its issues and challenges. International Journal or Advanced Research in Computer Science. 4:2, 305-311.

Clerc, Maurice. (2006). *Particle swarm optimization*. London, Newport Beach : ISTE.

Elbanna, A. & Sarker, S. (2016). The risks of agile software development: Learning from adopters. IEEE Software. 33:5, 72-79.

Hoda, R., Babb, J., & Norbjerg, J. (2013). Toward learning teams. *IEEE Software*. 30:4, 95-98.

Kennedy, James, & Russell C. Eberhart. (1995). Particle swarm optimization. *Neural Networks. IEEE International Conference on (Perth, Australia)*, 1942–48.

Kennedy, James, & Russell C. Eberhart. (1997). A Discrete binary version of the particle swarm algorithm. *Systems, Man, and Cybernetics. Computational Cybernetics and Simulation.* 5, 4104–4108.

Kovitz, B. (2003). Hidden skills that support phased and agile requirements engineering. *Requirements Engineering*. 8, 135-141.

Santos, V., Goldman, A., & de Souza, C.R.B. (2014). Fostering effective inter-team knowledge sharing in agile software development. Empirical Software Engineering. 20:4, 1004-1051.

Appendix: First-Third Increment Reports

OnTarget

A Course Scheduling Tool for College Students and Advisors

Member(s): Belinda Copus, Team #24 (document updated 3/21/2017)

1. Introduction

College students face many challenges transitioning to university life, including planning what courses to take each semester. Students are often unaware of the courses that need to be taken, what prerequisites are needed, and when these courses will be offered. It has been stated that academic advising is a vital link in retention (Drake, 2011). A logical conclusion could be made in that a student without a well- defined plan of study is less likely to complete a degree. Long range course offering schedules often are not available for students to access and create long range plans of study, independently, nor with any confidence that the courses they choose will be offered in the semester they believe it should be taken. Also, it is increasingly likely that a university will offer the same course on more than one campus and students often wish to take classes on more than one campus. Students must rely on a faculty or academic advisor to assist them in planning their coursework, but few students actually regularly meet with their advisor. Advisors also find the task of drafting a plan of study for the student to be quite tedious and time consuming. For an advisor to create an optimal plan that accounts for the many variables involved, is nearly impossible. Often, students have unique situations and constraints, adding even more complexity to the task. A software application that automates this effort is needed by both students and advisors.

2. Project Goal and Objectives (revised)

2.1 Overall Goal

OnTarget is an application that will facilitate the task of creating a student's college course plan. There is an additional side benefits to the departmental administrators, in that, a sense of future course demand can be determined by this application.

2.2 Significance/Uniqueness

There has been great effort made in creating automated advising tools, similar to this proposed system. There have been systems which are prescriptive, in that the student is provided a plan of study with little input. Others have created web-based decision support tools for academic advising is based on promoting engagement between the advisor and student (Feghali, Zbib, & Hallal, 2011). The system proposed is not necessarily unique in capabilities, but will be offered free and openly for those who would like to use the application. This system is superior to many readily available systems, in that it is able to consider distant future course offerings, whereas most currently marketed systems only consider a scheduling from a one-semester look ahead.

2.3 Objectives

This system is vital to saving time and providing a more accurate plan of study than what a person is able to create. It is the objective to develop and deliver this system for use by Fall 2017. While this proposed system is intended to be used by Computer Science students and faculty at a small university, it is possible that the application could be extended and adapted to other schools and degree programs.

2.4 System Features

The system will utilize the following data:

- Course catalog major requirements and general education requirements,
- Student's completed coursework and major/concentration,
- Course ranking per course in terms of time requirement outside of class,
- Course offering 5-year rolling plan, and
- Pre-requisite mapping.

This list comprises **the Foundational Data** used by the system.

A student will be able to generate a complete plan of study that leads to graduation. The student will be able to indicate which campus they prefer to take courses, the number of hours they wish to undertake during a semester, or whether they would like to take classes during the summer. General education requirements will be scheduled in addition to the requirements specified by the major. The student will also be able to specify load balance, in that the student can choose to balance course load between demanding courses and less demanding course. The plan will be stored so that it can be accessed by the student and/or advisor at some later time and by different hardware platforms.

The system will notify the advisor and student if a change is made in the 5-year rolling plan or in a pre-requisite requirement that will impact the current plan. Additionally, if the student is unable to obtain a sufficient grade in a course, the student and advisor will be notified and the plan can be automatically revised. If a student does not follow a plan for a particular semester, the system will automatically adjust the plan to reflect the shortfall.

The system will also track the number of students expected to take a particular course during a given semester to assist in allocating resources.

This system will web-based and be able to be utilized on multiple platforms, i.e. desktop, Android, and iOS platforms.

2.4.1 Specific Features

There are three primary types of users(actors) of this application. These include:

1) **Administrator** who are responsible for entering and maintaining the data used as input to the application,
2) **Advisor** who will review and modify student plans as needed, and
3) **Student** who will create a plan of study using the application.

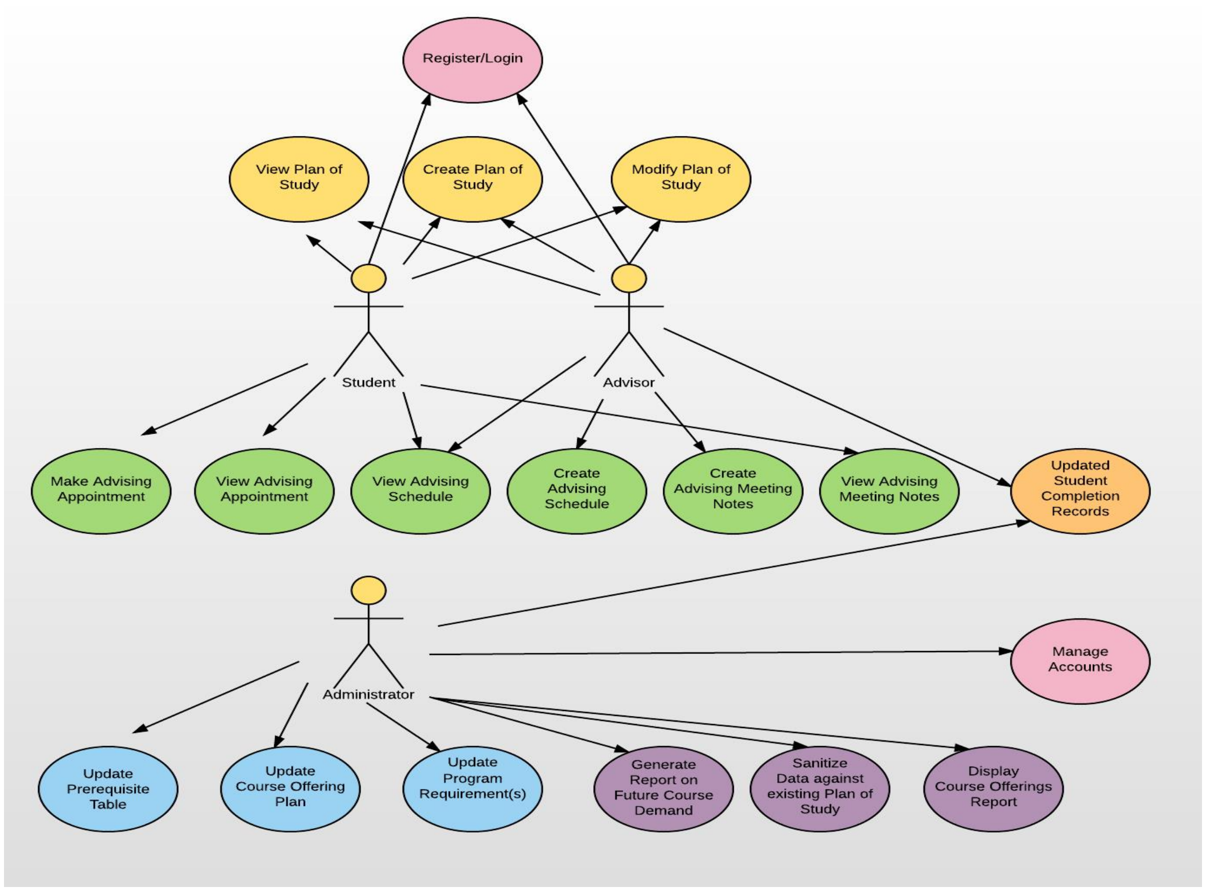
Specific features of the application include:

1. The Administrator will be able to populate/modify data used as input (foundational data) to application, including:
   a) long range course schedule table,
   b) prerequisite lists, program requirements,
   c) initial student records entry,
   d) storage and maintenance of student plans created by the application.
2. The Administrator will also manage the accounts of users who have access to the application.
3. The Administrator will run sanity checks to verify that foundational data and created plans are still congruent.
4. The Administrator will be able to run a report showing the course demand for future semesters.
5. The Advisor will be able to:
   a) View a plan for a specific student
   b) Modify a plan for a specific student
   c) Schedule advising appointments
   d) Be notified if changes to foundational data impact any created student plans.

6. The Student will be able to specify information regarding campus preference, number of hours per semester and generate a plan of study.

7. The Student will be notified if the generated plan needs to be modified for reasons, such as, course offering schedule has changed, prerequisite course has changed or was not obtained.
8. The Student with visual impairment will have the option of having their created plan spoken to them, removing the need for the student to visually review their plan.

Use Case Diagram for Plan of Study System


Register/Login
View Plan of Study
Create Plan of Study
Modify Plan of Study
Student
Advisor
Make Advising Appointment
View Advising Appointment
View Advising Schedule
Create Advising Schedule
Create Advising Meeting Notes
View Advising Meeting Notes
Updated Student Completion Records
Administrator
Manage Accounts
Update Prerequisite Table
Update Course Offering Plan
Update Program Requirement(s)
Generate Report on Future Course Demand
Sanitize Data against existing Plan of Study
Display Course Offerings Report

Use Case Descriptions

Manage Accounts

General case for managing accounts. Cf. Add Account, Delete Account, Edit Account.

Update Course Offering Plan

Goal: Modify/Update course offerings to reflect the departmental schedule of course offerings. This schedule is a 5-year rolling plan that tracks course offerings over three semesters during an a academic year, i.e. Fall, Spring, Summer.

Preconditions: Changes to the departmental schedule has been made.

Successful End Condition: Course offering additions and deletions are reflected.

Failed End Condition: none.

Primary Actor: Administrator

Secondary Actor: none.

Update Prerequisite Table

Goal: Modify/Update prerequisite list to reflect changes in course prerequisites. Prerequisite changes impact all courses, regardless of academic catalog.

Preconditions: Changes to the prerequisite has been made by the department.

Successful End Condition: Prerequisite additions and deletions are reflected.

Failed End Condition: none.

Primary Actor: Administrator

Secondary Actor: none.

Create Program Requirements

Goal: Every academic year program requirements are adjusted. A new program entry will be made every academic year for the programs used by this system. This system will be tracking students under four academic catalogs and six programs under each catalog.

Preconditions: A new course catalog has been approved for the next academic year.

Successful End Condition: Course offering additions and deletions are reflected.

Failed End Condition: none.

Primary Actor: Administrator

Secondary Actor: none.

Update Course Offerings

Goal: Every academic year new courses are added. A new course entry will be made every academic year for the programs used by this system.

Preconditions: A new course catalog has been approved for the next academic year.

Successful End Condition: Course offering additions and deletions are reflected.

Failed End Condition: none.

Primary Actor: Administrator

Secondary Actor: none.

Update Program Requirements

Goal: Modify/Update degree requirements to reflect the departmental schedule of course offerings. This schedule is a 5-year rolling plan that tracks course offerings over three semesters during an a academic year, i.e. Fall, Spring, Summer.

Preconditions: Changes to an existing program has been made.

Successful End Condition: Program additions and deletions are reflected.

Failed End Condition: none.

Primary Actor: Administrator

Secondary Actor: none.

Sanitize Foundational Data Against Existing Plan of Studies

Goal: As changes are made in the foundational data – course offering schedule, program requirements, prerequisites, and student course completion, current Plan of Study could be impacted. There is a need to compare any changes to the plans and notify student and advisor of possible changes and/or issues.

Preconditions: Changes to foundational data has been made.

Successful End Condition: A list of students and courses are created for each potential issue discovered, if any.

Failed End Condition: none.

Primary Actor: Administrator

Secondary Actor: Student and Advisor.

Generate Report on Future Course Demand

Goal: Generate report displaying the number of students intending to take a course, by course and semester.

Preconditions: none.

Successful End Condition: Report is created.

Failed End Condition: none.

Primary Actor: Administrator

Secondary Actor: none.

Display Course Offerings Report

Goal: Generate a report that will display when a course is planned to be offered, by course and semester. The report should also be able to display the courses offered in a particular semester, or over the entire 5-year period.

Preconditions: Request must contain the course prefix to display a single course offering. A semester will be provided to display the courses in a single semester, and an option will be selected to display the entire 5-year period.

Successful End Condition: Report is created.

Failed End Condition: none.

Primary Actor: Administrator

Secondary Actor: none. Although providing this functionality to the Advisor and Student should be considered for a future revision.

Register/Login

Goal: Allow a new user to register an account. Allow an existing user to log into their account. A user can be a student, advisor, or administrator.

Preconditions: To register the user must already be included in the list of students. The student will provide a 700# student id number and password. The advisor will provide a 700# identification number and a password. To login, the student, advisor, or administrator will provide their 700# for a login name and their password for verification.

Successful End Condition: For registration – account is created. For login – user is allowed access to the system or access is denied.

Failed End Condition: none.

Primary Actor: Administrator, Advisor, and Student

Secondary Actor: none.

Create a Plan of Study

Goal: A plan of study is generated for the student based on the foundational data.

Preconditions: Student must be included in the list of students.

Successful End Condition: Report for a plan of study is displayed to the user and saved to the system.

Failed End Condition: none.

Primary Actor: Advisor and Student

Secondary Actor: none.

View Plan of Study

Goal: A plan of study is displayed based on a student 700#.

Preconditions: Student must have a plan of study previously created.

Successful End Condition: Report for a plan of study is displayed to the user. If no plan for the student exists, the user is notified of this condition.

Failed End Condition: none.

Primary Actor: Advisor and Student

Secondary Actor: none.

Modify Plan of Study

Goal: A course in a plan of study is modified based on a student 700#. This will allow courses to be manually moved to a different semester or to allow for the occasional exception to a course requirement.

The user will identify the course to remove or move, the originating semester and destination semester, and a course number that will replace, if needed.

Preconditions: Student must have a plan of study previously created.

Successful End Condition: Report for a plan of study is displayed to the user. If no plan for the student exists, the user is notified of this condition. Updated plan is saved to the system.

Failed End Condition: none.

Primary Actor: Advisor and Student

Secondary Actor: none.

Make Advising Appointment

Goal: Students need to attend advising as often as every semester. Appointments can be made through this interface that will update the calendars of individuals involved.

Preconditions: Student must be listed in the list of students and be registered.

Successful End Condition: Appointment is scheduled and appears on the Student and Advisor calendar. The Advisor may choose to schedule the student, and if the Student is not a registered user, message appears to the Advisor.

Failed End Condition: none.

Primary Actor: Advisor and Student

Secondary Actor: none.

View Advising Appointment

Goal: Students need to attend advising as often as every semester. Scheduled Appointments can be viewed through this interface.

Preconditions: Student must be listed in the list of students and be registered.

Successful End Condition: Appointment schedule is displayed to the Advisor or Student. The Advisor may choose to schedule of a single student, and if the Student is not a registered user, message appears to the Advisor.

Failed End Condition: none.

Primary Actor: Advisor and Student

Secondary Actor: none.

Create Advising Schedule

Goal: Students need to attend advising as often as every semester. The Advisor must create a list of times in which the student can select to attend advising.

Preconditions: Student must be listed in the list of students and be registered. The Advisor must be a registered user of the system.

Successful End Condition: Appointment schedule is created and available to the Student and Advisor.

Failed End Condition: none.

Primary Actor: Advisor and Student

Secondary Actor: none.

View Advising Schedule

Goal: Students need to attend advising as often as every semester. Scheduled Appointments can be viewed through this interface. Advisors can view their entire schedule.

Preconditions: Student must be listed in the list of students and be registered. Advisor must also be a registered user of the system.

Successful End Condition: Appointment schedule is displayed to the Advisor. The Advisor may choose to view a schedule for a day, week, or month time period.

Failed End Condition: none.

Primary Actor: Advisor and Student

Secondary Actor: none.

Add Advising Meeting Notes

Goal: Advisors will need to maintain notes from advising meeting with a student.

Preconditions: Student must be listed in the list of students and be registered. Advisor must also be a registered user of the system.

Successful End Condition: Advising notes are stored to the system and associated with a plan of study.

Failed End Condition: none.

Primary Actor: Advisor

Secondary Actor: none.

View Advising Meeting Notes

Goal: Students and Advisors will need to refer to notes from an advising meeting.

Preconditions: Student must be listed in the list of students and be registered. Advisor must also be a registered user of the system.

Successful End Condition: Advising notes are displayed to the user.

Failed End Condition: none.

Primary Actor: Advisor and Student

Secondary Actor: none.

Update Student Completion Records

Goal: Student course completion data is vital to this system. At this time, it must be manually updated at the end of a semester. Courses that are successfully completed by the student are added to the Students completed coursework list.

Preconditions: Student must be listed in the list of students and be registered. Advisor must also be a registered user of the system.

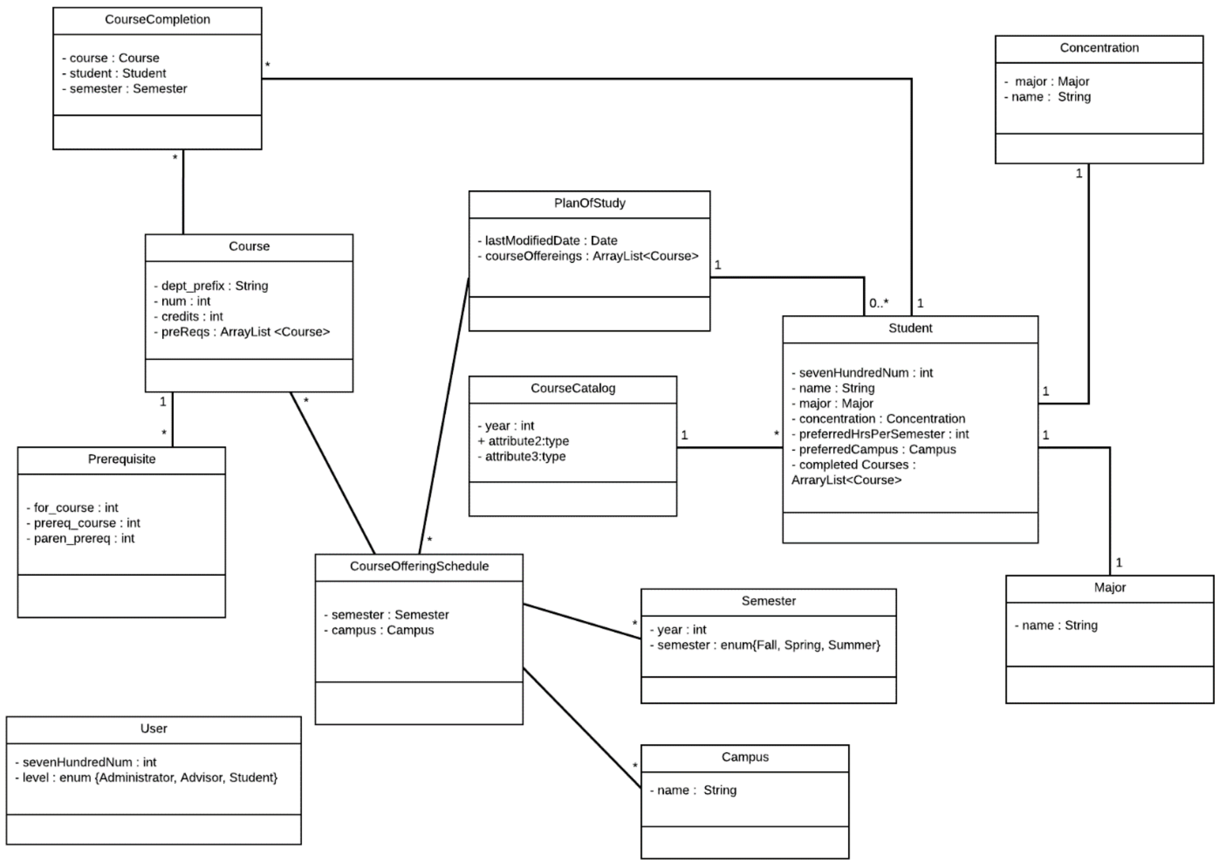
Successful End Condition: Data is updated appropriately.

Failed End Condition: none.

Primary Actor: Advisor

Secondary Actor: none.

Class Diagram for Plan of Study System



Schema for Plan of Study System Database

```
drop database if exists ontarget;
create database ontarget;
use ontarget;


create table Course (
  course_id int unsigned not null auto_increment,
  dept_prefix varchar(6) not null,
  course_num smallint not null,
  course_name varchar(50) not null,
  credit_hours smallint unsigned not null,
  primary key (course_id),
  index(course_id),
  index(dept_prefix, course_num)
);

create table Prerequisite (
  prerequisite_id int unsigned not null auto_increment,
  for_course int unsigned default null,    #fk Course.course_id
  prereq_course int unsigned default null, #fk Course.course_id
  parent_prereq int unsigned default null, #fk Prerequisite.prerequisite_id
  primary key (prerequisite_id),
  index(prerequisite_id),
  index(for_course)
);

create table CourseCompletion (
  completion_id int unsigned not null auto_increment,
  course int unsigned not null,   #fk
  student int unsigned not null,  #fk
  semester int unsigned not null, #fk
  primary key (completion_id),
  index(completion_id),
  index(student)
);

create table Major (
  major_id int unsigned not null auto_increment,
  name varchar(50) not null,
  primary key (major_id),
  index(major_id)
);

create table Concentration (
  concentration_id int unsigned not null auto_increment,
  name varchar(50) not null,
  major int unsigned not null, #fk
  primary key (concentration_id),
  index(concentration_id)
);

create table Campus (
  campus_id int unsigned not null auto_increment,
  name varchar(50) not null,
  primary key (campus_id),
  index(campus_id)
);

create table PlanOfStudy (
  pos_id int unsigned not null auto_increment,
  dateLastModified datetime default null,
  student int unsigned not null, #fk
  primary key (pos_id),
  index(pos_id),
  index(student)
);

create table Student (
  student_id int unsigned not null,
  l_name varchar(50) not null,
  f_name varchar(50) not null,
  major int unsigned not null, #fk
  concentration int unsigned not null, #fk
  preferred_load smallint default null,
  preferred_campus int unsigned default null, #fk
  primary key (student_id),
  index(student_id),
  index(l_name),
  index(f_name),
  index(l_name, f_name)
);

create table Semester (
  semester_id int unsigned not null auto_increment,
  academic_year smallint not null,
  semester_code smallint not null,
  primary key (semester_id),
  index(semester_id)
);

create table CourseOffering (
  offering_id int unsigned not null auto_increment,
  course int unsigned not null, #fk
  semester int unsigned not null, #fk
  campus int unsigned not null, #fk
  primary key (offering_id),
  index(offering_id),
  index(semester),
  index(campus)
);

create table CourseCatalog (
  catalog_id int unsigned not null auto_increment,
  catalog_year smallint not null,
  concentration int unsigned not null, #fk
  primary key (catalog_id),
  index(catalog_id),
  index(concentration)
);

create table CourseGroup (
  group_id int unsigned not null auto_increment,
  name varchar(50) not null,
  catalog_id int unsigned not null, #fk CourseCatalog.catalog_id
  parent_id int unsigned default null, #fk CourseGroup.group_id
  minimum_hours smallint unsigned not null,
  primary key (group_id),
  index(group_id),
  index(catalog_id),
  index(parent_id)
);

create table CourseMapCourseGroup (
  id int unsigned not null auto_increment,
  course_id int unsigned not null, #fk Course.course_id
  course_group_id int unsigned not null, #fk CourseGroup.group_id
  primary key (id),
  index (id),
  index (course_id),
  index (course_group_id)
);

create table DegreeRequirement (
  requirement_id int unsigned not null auto_increment,
  catalog_id int unsigned not null, #fk CourseCatalog.catalog_id
  concentration_id int unsigned not null, #fk Concentration.concentration_id
  course_group_id int unsigned not null, #fk CourseGroup.group_id
  primary key (requirement_id),
  index (requirement_id),
  index (catalog_id),
  index (concentration_id),
  index (course_group_id)
);


alter table Prerequisite
  add foreign key(for_course)
  references Course(course_id);

alter table Prerequisite
  add foreign key(prereq_course)
  references Course(course_id);

alter table Prerequisite
```

```
  add foreign key(parent_prereq)
  references Prerequisite(prerequisite_id);

alter table CourseCompletion
  add foreign key(course)
  references Course(course_id);

alter table CourseCompletion
  add foreign key(student)
  references Student(student_id);

alter table CourseCompletion
  add foreign key(semester)
  references Semester(semester_id);

alter table Concentration
  add foreign key(major)
  references Major(major_id);

alter table PlanOfStudy
  add foreign key(student)
  references Student(student_id);

alter table Student
  add foreign key(major)
  references Major(major_id);

alter table Student
  add foreign key(concentration)
  references Concentration(concentration_id);

alter table Student
  add foreign key(preferred_campus)
  references Campus(campus_id);

alter table CourseOffering
  add foreign key(course)
  references Course(course_id);

alter table CourseOffering
  add foreign key(semester)
  references Semester(semester_id);

alter table CourseOffering
  add foreign key(campus)
  references Campus(campus_id);
```

```
alter table CourseCatalog
  add foreign key(concentration)
  references Concentration(concentration_id);

alter table CourseGroup
  add foreign key(catalog_id)
  references CourseCatalog(catalog_id);

alter table CourseGroup
  add foreign key(parent_id)
  references CourseGroup(group_id);

alter table DegreeRequirement
  add foreign key(catalog_id)
  references CourseCatalog(catalog_id);

alter table DegreeRequirement
  add foreign key(concentration_id)
  references Concentration(concentration_id);

alter table DegreeRequirement
  add foreign key(course_group_id)
  references CourseGroup(group_id);

alter table CourseMapCourseGroup
  add foreign key(course_group_id)
  references CourseGroup(group_id);

alter table CourseMapCourseGroup
  add foreign key(course_id)
  references Course(course_id);
```

3. Project Plan

3.1 Schedule

Increment 1 (Due on 2/17/2017)

1. Draft Use cases
2. Draft Scenarios/Use Case Stories
3. Develop schema for the databases: course offering schedule, prerequisite list, student completed coursework, programs of study, student plans, courses
4. Design User Interface for entering data into the database
5. Implement User Interface for entering data into the database
6. Populate databases (course offering schedule, prerequisite list, student completed coursework, programs of study, and courses) with data using user interface
7. Test item 5 from this increment.

Increment 2 (Due on 3/10/2017)

1. Develop Student Page that allows student to specify campus and desired hours and button to create a plan.
2. Develop Student Page that allows student to retrieve their plan.
3. Develop Advisor Page that allows advisor to view a student's plan of study.
4. Develop Advisor Page that allows advisor to modify a student's plan of study.
5. Create Launch Page
6. Create Login Page
7. Create Registration Page
8. Test each item in increment 1 & 2
9. Update use cases and scenarios, if needed.

Increment 3 (Due on 4/7/2017)

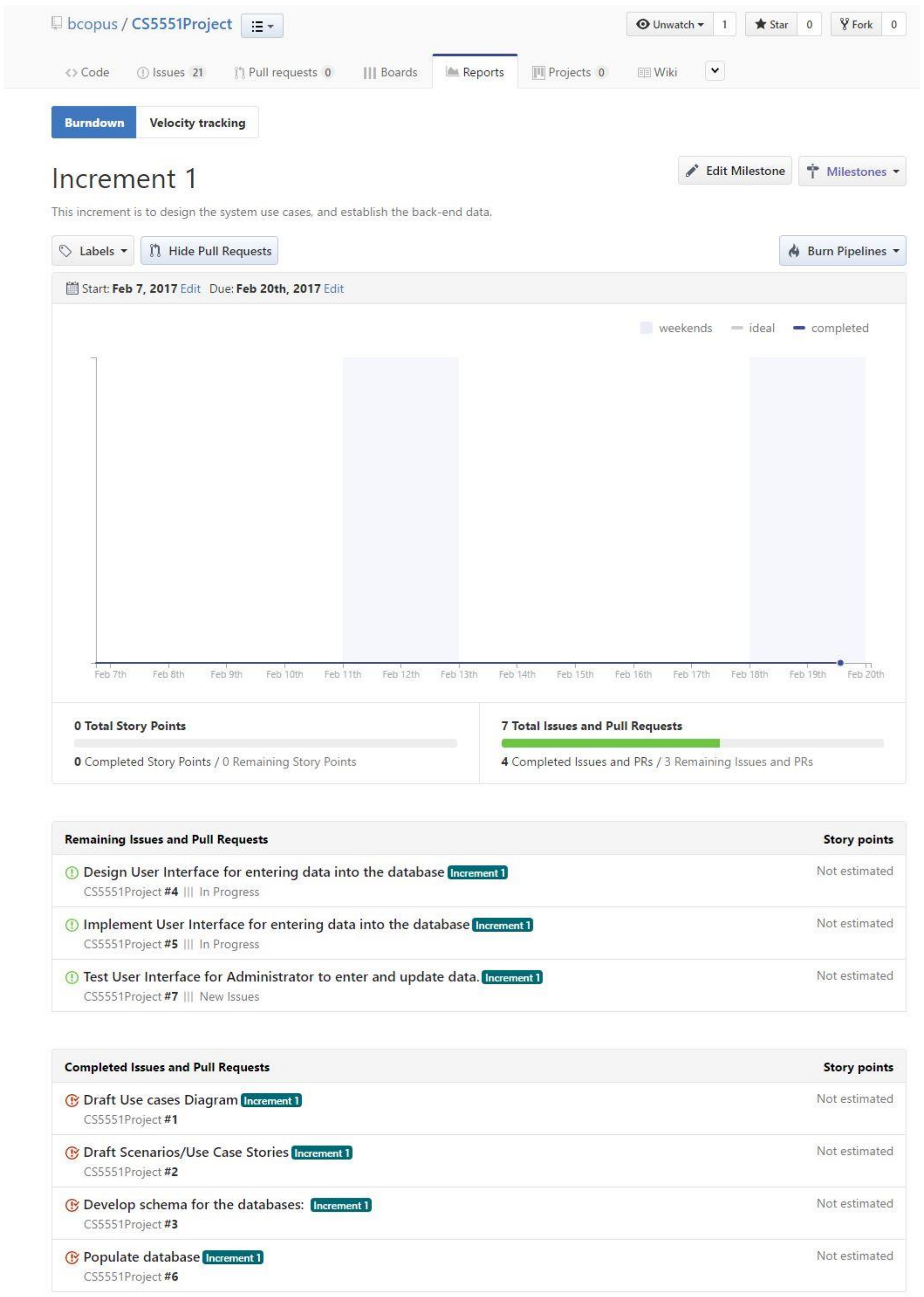
1. Add functionality to administrator check student plans if a change has been made in foundational data. This will also include notifying the advisor and student regarding potential issue.
2. Add functionality to student page and advisor page that allows for manual modification of a created plan.
3. Add functionality that will speak a student plan to the student.
4. Test each item in increment 1-3
5. Update use cases and scenarios, if needed.

Increment 4 (Due on 4/26/2017)

1. Develop report function for the administrator that displays demand for an upcoming course
2. Draft deployment plan
3. Final testing
4. Update use cases and scenarios, if needed.

3.2 Project Timelines, Members, Task Responsibility

3.3 Burndown Chart



Increment 1

This increment is to design the system use cases, and establish the back-end data.

Start: **Feb 7, 2017** Edit Due: **Feb 20th, 2017** Edit

0 Total Story Points

0 Completed Story Points / 0 Remaining Story Points

7 Total Issues and Pull Requests

4 Completed Issues and PRs / 3 Remaining Issues and PRs

Remaining Issues and Pull Requests	Story points
Design User Interface for entering data into the database Increment 1 CS5551Project **#4** ||| In Progress	Not estimated
Implement User Interface for entering data into the database Increment 1 CS5551Project **#5** ||| In Progress	Not estimated
Test User Interface for Administrator to enter and update data. Increment 1 CS5551Project **#7** ||| New Issues	Not estimated

Completed Issues and Pull Requests	Story points
Draft Use cases Diagram Increment 1 CS5551Project **#1**	Not estimated
Draft Scenarios/Use Case Stories Increment 1 CS5551Project **#2**	Not estimated
Develop schema for the databases: Increment 1 CS5551Project **#3**	Not estimated
Populate database Increment 1 CS5551Project **#6**	Not estimated

4. First Increment Report

4.1 Existing Services/REST API

No services were used in this increment. API service of text to voice will be used in Increment 3.

4.2 Detail Design of Features

Please refer to section 2.4

4.3 Testing

N/A in this increment

4.4 Implementation (using Android/Angular.js/Bootstrap)

N/A in this increment

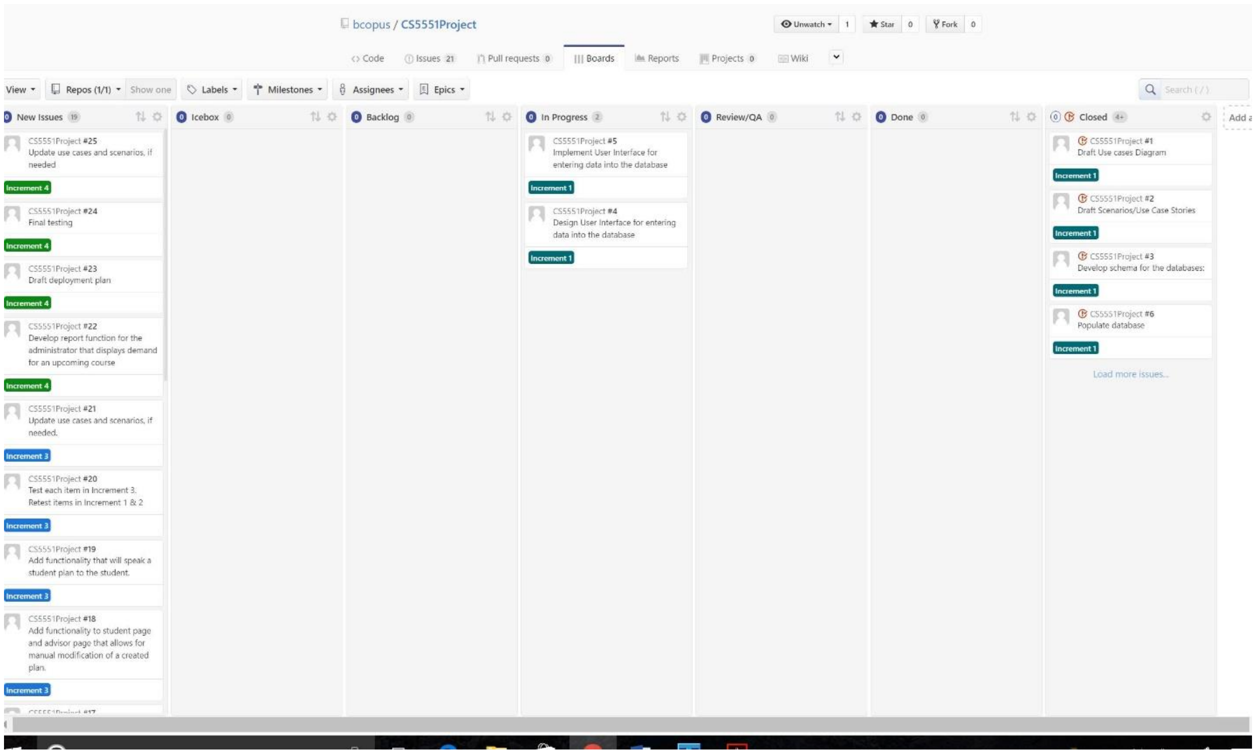
4.5 Deployment

N/A in this increment

4.6 Project Management

Please see section on Burndown Chart.

Increments and Issues entered into ZenHub.



New Issues 19

CS5551Project #25
Update use cases and scenarios, if needed
Increment 4

CS5551Project #24
Final testing
Increment 4

CS5551Project #23
Draft deployment plan
Increment 4

CS5551Project #22
Develop report function for the administrator that displays demand for an upcoming course
Increment 4

CS5551Project #21
Update use cases and scenarios, if needed.
Increment 3

CS5551Project #20
Test each item in Increment 3. Retest items in Increment 1 & 2
Increment 3

CS5551Project #19
Add functionality that will speak a student plan to the student.
Increment 3

CS5551Project #18
Add functionality to student page and advisor page that allows for manual modification of a created plan.
Increment 3

Icebox 0

Backlog 0

In Progress 2

CS5551Project #5
Implement User Interface for entering data into the database
Increment 1

CS5551Project #4
Design User Interface for entering data into the database
Increment 1

Review/QA 0

Done 0

Closed 4+

CS5551Project #1
Draft Use cases Diagram
Increment 1

CS5551Project #2
Draft Scenarios/Use Case Stories
Increment 1

CS5551Project #3
Develop schema for the databases:
Increment 1

CS5551Project #6
Populate database
Increment 1

Load more issues...

5. Second Increment Report

5.1 Existing Services/REST API

No services were used in this increment. API service of text to voice will be used in Increment 3.

5.2 Detail Design of Features

Please refer to section 2.4

5.3 Testing

N/A in this increment

5.4 Implementation (using Android/Angular.js/Bootstrap)

Many implementation changes/decisions were made in this increment. These changes were made because I became aware of tools and how to apply certain tools to this project. Specifically,

1. Client Side – learned to use Ionic framework.
2. Developed basic client app. For the student to view plan
3. Server side researched server side frameworks based on NodeJS
4. Found Feathers – a thin wrapper around Express and Socket.io. Feathers was good because the documentation to get started was really straight forward. If Feathers is determined to add no real value, consideration to move solely to Express will be considered.
5. Since the application doesn't need real-time web sockets, but only a REST interface, not using Socket.io functionality, but Feathers is just as happy providing a REST interface. Feathers seemed a lot easier to use than Express alone for a beginner. Feathers has lots of DB connection options, including MongoDB, MySQL, neDB, and others.
6. Decided to stay with MySQL database because I am most familiar with this database. Even though the back end framework provides wrappers around the db, still hve to produce appropriate queries and how data is stored. MongoDB was considered but rejected because it wasn't going to really add anything that wasn't already being provided by MySQL for this particular application.

Prototype app that exposes a REST api and client connects to it and is able to receive data. My client can communicate with a server. Created a new instance of the server that is built around a MySQL database and it is able to connect to the database.

5.5 Future Steps for Iteration 3

Now that the architecture is understood, data needs to be created so that features and functionality can be added to the application.

The following are action items for Interation 3:

1. Server side – build API so that client (student and advisor) can get real data. Add functionality to the server that allows data to be reset to a known state in the database.
2. Client side – Auto generate student plan based on cost function: factors such semester load in terms of hours or difficulty, campus are considered, etc.
3. Allow courses to be moved manually.
4. Advising appointment functionality – initiate communication between student and advisor.

5.6 Deployment

N/A in this increment

5.7 Project Management

Please see section on Burndown Chart.

Increments and Issues entered into ZenHub.

6. Third Increment Report

6.1 Existing Services/REST API

Two APIs were created in this increment:

1. Student/Advisor functionality to read and display student data to user.
2. Plan of Study functionality which includes CRUD operations

6.2 Detail Design of Features

Please refer to section 2.4

6.3 Testing

N/A in this increment

6.4 Implementation (using Android/Angular.js/Bootstrap)

Began implementation of AI scheduler for producing plans of study. The AI Scheduler will utilize a variation of discrete particle swarm optimization (dPSO). This implementation optimizes a cost function that consider user preferences with respect to campus, coarse load, and target graduation date. Only plans that obey the constraints, e.g. course prerequisites, required hours in select-group electives, etc. will be chosen.

This intelligent scheduler is being implemented in Java, is not quite complete, but currently is about two-thousand lines of code. The AI Scheduler will be deployed as a service in a web container for use by the NodeJS/Express – based on the backend.

Currently, integration with the MySQL database is being done so that the tool can gather constraints before running the optimization algorithm.

Other items which were added in increment three, include user interface. The user can specify

7. Future Steps for Iteration 4

Efforts began in Iteration 3 will be complete during the fourth iteration. Additionally, the following items also will need to be addressed in the fourth iteration:

1. Allow courses to be moved manually.

2. Advising appointment functionality – initiate communication between student and advisor. Integrate Google calendar to allow for easy appointment scheduling and management.

8. Deployment

N/A in this increment

9. Project Management

Please see section on Burndown Chart. Increments and Issues entered into ZenHub.

Bibliography

Drake, J. (2011). The role of academic advising in student retention and persistence. About Campus. 16(3), 8-12.

Feghali, T., Zbib, I., & Hallal, S. (2011). A web-based decision support tool for academic advising. Educational Technology & Society. 14(1), 82-94.

Colleague Student Planning http://www.ellucian.com/Software/Colleague-Student-Planning/

Smart Planner http://www.tbginc.com/smart-planner/

ONTARGET

AN INTELLIGENT PLAN OF STUDY TOOL

Belinda J. Copus

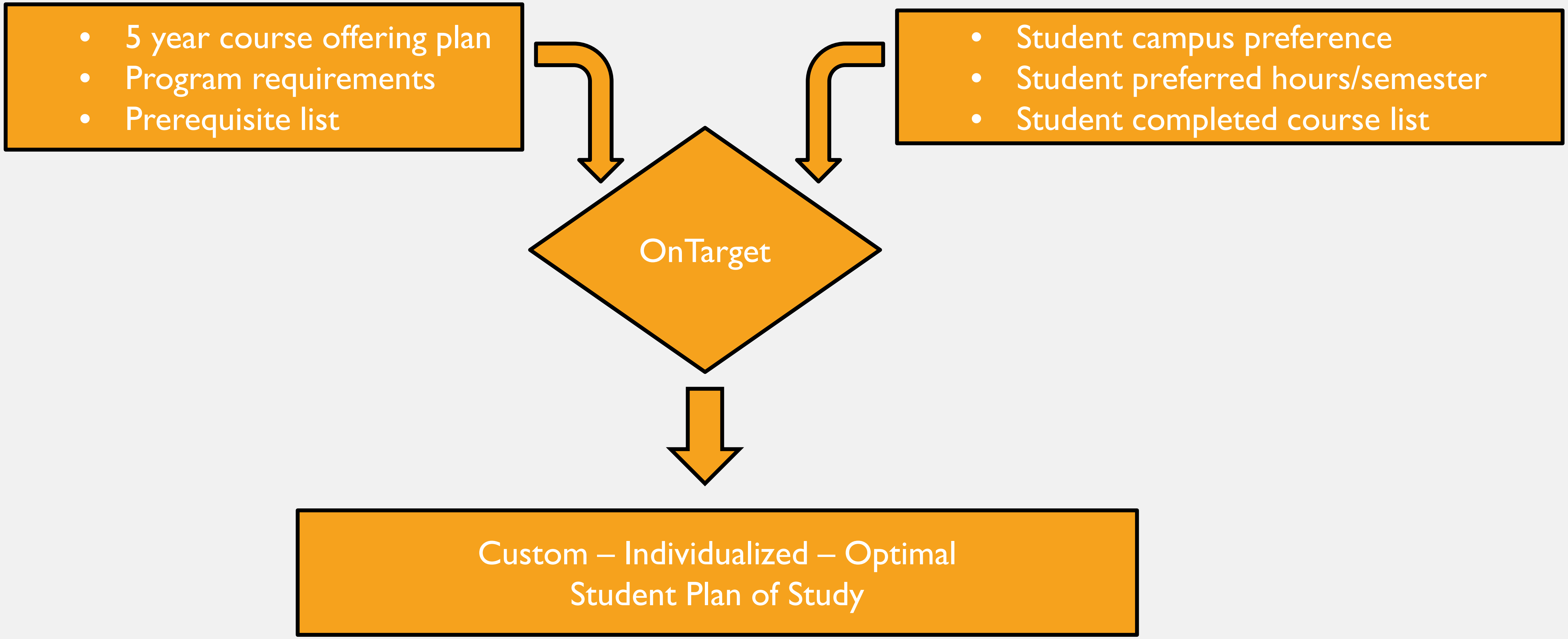
Team #24

CS 5551 Spring 2017

THE PROBLEM

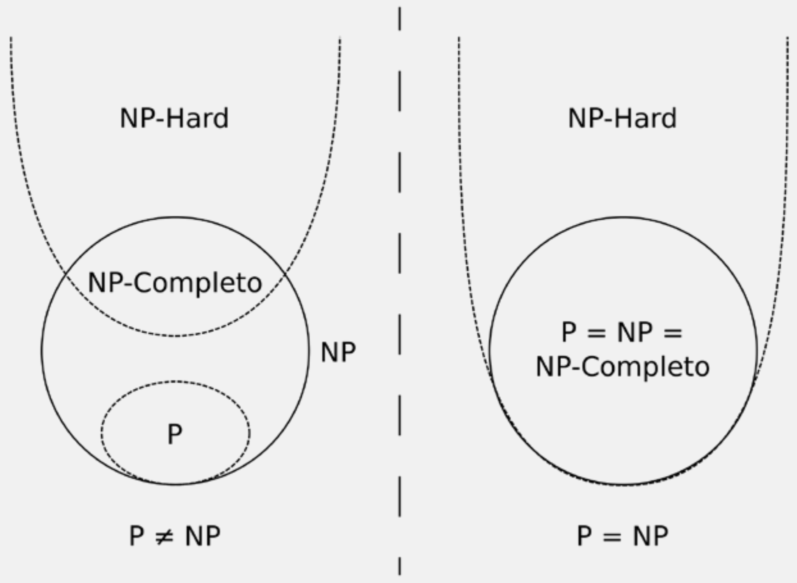
- ***College students need a plan of study to successfully reach graduation.***
- ***Academic Advisors often are tasked with creating a plan for a student, which can take 30 minutes to an hour per student.***
- ***From my own experience, I advise 60 students a semester, under 4 different catalogs, 7 programs, and with course offerings on two campuses. This activity gets complicated fast!***
- ***OnTarget is the answer to automating this task.***

ONTARGET OVERVIEW


• 5 year course offering plan
• Program requirements
• Prerequisite list
• Student campus preference
• Student preferred hours/semester
• Student completed course list
OnTarget
Custom – Individualized – Optimal
Student Plan of Study

THE TECHNICAL CHALLENGE

- Find an optimal path to graduation
- Consider:
  - Catalog
  - Degree
  - Concentration
  - Prior Coursework
  - Campus preference
  - Course loading
  - Prerequisites
- Similar complexity to constrained asymmetric TSP



$O(n!)$

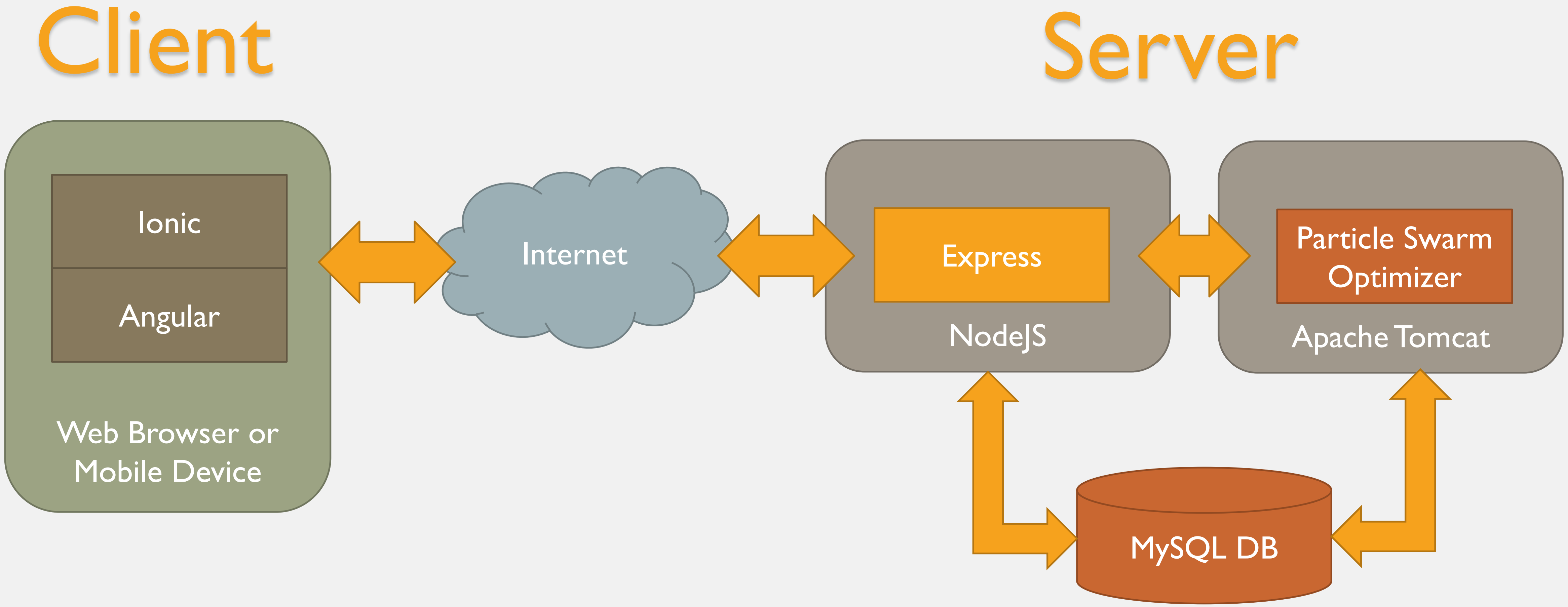
PARTICLE SWARM OPTIMIZER

- Particle Swarm Optimization (PSO) is a metaheuristic algorithm
- Belongs to the class of "Scruffy" Artificial Intelligence
- Ideal for high-dimension problem spaces with complex constraints
- Much faster than brute-force algorithms, which are O(n!)
- Does not guarantee optimal solution, but usually very close

USERS AND USE CASES

Action	Advisor	Student
View a plan	◆	◆
Create a plan	◆	
Modify a plan	◆	
Update courses completed by student	◆	
Request appointment with student/advisor	◆	◆
Update preferences		◆

ARCHITECTURE



REFERENCES / CREDITS

- PARTICLE SWARM OPTIMIZATION
  - Kennedy, James, and Russell C. Eberhart. "Particle Swarm Optimization." Neural Networks, 1995. IEEE International Conference on (Perth, Australia), 1995, 1942–48.
  - Kennedy, James, and Russell C. Eberhart. "A Discrete Binary Version of the Particle Swarm Algorithm." In Systems, Man, and Cybernetics, 1997. Computational Cybernetics and Simulation., 1997 IEEE International Conference on, 5:4104–8. IEEE, 1997.
  - Clerc, Maurice. Particle Swarm Optimization. London, Newport Beach: ISTE, 2006.
- FRAMEWORKS
  - https://ionicframework.com/
  - https://expressjs.com/
  - https://nodejs.org/en/
- IMAGES
  - https://commons.wikimedia.org/wiki/File:P_np_np-completo_np-hard.svg

This work was done in partial fulfillment of the requirements of CS5551: Advanced Software Engineering, CSEE Department, University of Missouri – Kansas City (Spring 2017). Instructor: Dr. Yugyung Lee, TAs: Arunit Gupta, Marmik Patel, Ram Gopal Mangena, Sidrah Junaid