RIT Department of Software Engineering and Department of Mechanical Engineering

Online Program Flowchart, Advising, and Scheduling System

Team Flowriders

Andrew Bona  Jeff Kelley  Sean Madden
Joshua Peterson  Bryan Schick

Project Sponsor

Dr. James Vallino, Dr. Ed Hensel, Lana Verchage, Sarah Mittiga

Faculty Coach

Daniel Krutz

Project Overview

The Flowriders’ Mechanical Engineering and Software Engineering Academic Advising System (MeSe Advising) is a system to aid in student record keeping, schedule planning, and schedule prediction. The program worksheet is the primary point of interface between a student and his or her academic advisor. Many programs have a graphic representation of the worksheet in flowchart form. Discussions about a student’s program of study revolve around completion of all the program requirements as specified on the worksheet or flowchart. Within a program, the record of the courses a student has taken is quite often maintained on a paper worksheet or perhaps in an electronic document or spreadsheet. Forward looking planning is usually captured with notes made on a paper document with the risk of lost paper and the need to make multiple copies of notes for the student and advisor. The conversion to a semester calendar will place a large burden on the advising staff. Every student on in the program becomes a transfer student requiring a complete analysis of the student’s quarter record and finding a best fit into the semester curriculum. Conversion also poses a scheduling challenge. Many years of past experience, which is embedded in the current schedule, captures the knowledge of when a program offers courses, and how many sections will meet student needs. A switch to a completely new curriculum and calendar system resets that knowledge to zero. The advising staff and program administrators would greatly benefit from a system that assists with both of these tasks through conversion and beyond.

The main problem is that student records need to be maintained by hand in each department, separate from the registrar. This 'by hand' system causes 2 problems. It creates a very difficult process for academic advisors to update each students record at the beginning of each quarter. This also means that if a student wants to view their record they need to request it from their academic advisor.

This project proposes to move the current advising system, which is primarily paper- or local file-based, to a web-based online system. By making these records accessible through the web, this system will allow the advising staff to have easier access to the students’ worksheets, along with all the data associated with a student’s program record that each program maintains. An online system will allow schedule planning and experimentation by the student and academic advisors, and save these scheduling plans for later use. During advising discussions, students and advisors will sometimes work from the worksheet and other times from the program flowchart.
This system can display a student’s program record in either form. If each student’s program flow is planned for at least the next year, the system will hold valuable course scheduling information that will be particularly important when we convert to a semester calendar. Finally, this system provides a high degree of end-user configurability to define courses, programs, and display formats.

**Basic Requirements**

**Flowchart View**

The Flowchart View is designed to be a visual representation of a student's record. It contains all courses that a student has completed and provides an area to plan for future courses. It also contains all placeholders for the curricula that are assigned to the student. The main focus of this view is to interact with the placeholders and show which aspects of the curricula have been completed based on the student's course history. As a whole, this view is of High priority because it provides an interface for many of the other required features of the program. It is also important to this feature to reflect the known flowchart to the student body because it is a system that many people already know how to read. We don’t want to re-teach everybody how to read our flowchart.

**Worksheet View**

The Worksheet View is designed to mirror the student's department worksheet. This form will automatically populate with the students record and allow them to plan out there future terms. This is a major feature of the project because it was expresses that a drawback of the current implementation of the worksheet is completely offline and must be maintained manually. Like the flowchart, it is important for this view to reflect the known system to the student’s worksheet.

**Technical Administrator Role**

The Technical Administrator Role will be a person that maintains the deployment of the software. They will deploy the software, maintain any backups, add users to the system, and maintain system health. They will also have to create new academic terms to the system. Since the system doesn’t know what date terms are starting and ending, these will have to be manually inserted into the system. This serves two purposes, first it ensures that a measure of time is kept in the system so we know whether courses were taken in the past, if they are current courses, or if they are planned, and second it prevents students from planning an excessive amount of time into the future which can over burden the system.

**Academic Advisor Role**

The Academic Advisor Role is a specific user role for the web application. These users are the people which advise the student body on what courses to take. They are unable to define curricula and also cannot view any statistics on courses. They can approve (aka bless) curricula that students have generated that marks it as approved for use by a student.
**Program Administrator Role**

The Program Administrator Role is one of the roles with the most abilities. This role can view and edit a students’ record within the system. They are also the person who creates the curriculum for the program. They will also be able to add new Academic Advisors to the system.

**Student Role**

The student role allows students to access the system, pull their student record, and plan for future terms at the university. The student is not allowed to view any other students’ record other than their own to adhere to FERPA laws. The student can also save any planned information on their student record in order to be used in scheduling requirements prediction.

**Constraints**

There were only 2 constraints given to us by the customer, one of which designated quite a large portion of our technologies. The first was that this application must be web based. Being web based guaranteed universal access by both people on and off campus including people on co-op. Our second constraint said that this system must be maintainable by ITS. This means that our system needs to be deployable on an ITS system and have appropriate documentation. Deploying to an ITS system dictated that our technologies be Java 1.6, Struts 2, Oracle 11g, and Linux.

**Development Process**

We decided to use an incremental development methodology. The main goal behind this development methodology is to compensate for weaknesses within the waterfall model where business value is delivered all at once at the end of the release cycle. By breaking the project into a series of well-defined releases, this process allows the development team to deliver value in smaller chunks while allowing a higher level of project visibility for the customer all while reducing risk. While the ideas behind iterative are a large component of other methodologies like the Rational Unified Process, Extreme Programming (XP) and several other agile processes, they are quite effective when applied by themselves.

Using this staged delivery methodology will also allow us to get early warnings of problems. We will plan to deliver releases early and often we will get frequent, indisputable progress reports back from our customers. Either the release is done on time or it's not. The increments work quality is obvious from the release's quality. If the development team is in trouble, we will discover that within one of the first increments rather than when 90% of the project is done.

Additionally, the structure of incremental development will allow for the relative inexperience of the five developers in our project to have some leeway within the scope of the project. We feel that the Waterfall methodology does not provide the requisite visibility and that we do not have the experience to make an agile process work. Iterative was a natural choice for this project.
Project Schedule: Planned and Actual

We generally built our schedule based on 2 factors. One was what the customer wanted in new features and the second was what underlying functionality was needed to build those new features. Using our incremental development process our initial release was to build the basic system that we could build off of. We defined the time line for each increment based on our previous experience with the system. This made our first estimate very hard to make which caused us to miss our deadline by a few days. Once we had our baseline to work off of we were able to better estimate our time and didn't miss any other deadlines.

Through the entire project we had 5 releases ranging from 2-3 weeks in development time. The amount of time between releases was determined by the amount of work we estimated we had. While the total number of tasks we assigned for each release increased each release, we felt this was because we increasingly become better with the technologies we were using and have a better and more stable system to work off of. Since we set our own deadlines and the customer had no deadlines that we had to meet there was nothing that we had to adapt to in order to meet a deadline. If a feature in a release relied upon the completion of another we would set an internal team due date so we could still have equal development time for each feature.

System Design

Struts 2 Architecture

To start our application is a Struts 2 application. In order for our some of our design choices to make sense we must first cover some basics of the Struts 2 architecture. The goal of Struts is to separate the model, where the application logic that interacts with a database exists, from the view, the HTML pages, and the controller, the part the binds information between the views and model. Struts’ provides the controller and facilitates the writing of templates for the view or presentation layer. We are responsible for writing the model code, and for creating a central configuration file that binds together model, view and controller.
You can see from the diagram above where our code exits in the Struts 2 ecosystem. Our action classes relate to the different feature points that exist in our system.

**Database Design**

Our database design is fairly large because of the many different objects that we need to keep track of and the complex relationships that exist between them.
To start we need to have a student in our database. That student needs to have a record of classes (AcaClass) that they have taken. A grade in the AcaClass denotes that the student has taken that class. If the AcaClass does not have a grade then we are assuming that is a planned course in the future. To know when the course was taken we have AcaTerms and AcaYears. These are actual points in time. AcaTerms are typically stored as something like ‘20071’ but are dynamic enough to represent the semester codes. The AcaYears also represent what academic year a class was taken. These AcaClasses fulfill courses. A course includes information, like the course number, description, and course name. We did this because there would be a lot of repeat of information if we stored all the same information in each student who took the same course.
Going back to the student, a student can have zero to many curriculums assigned to them. Using our definition of curriculum we can then also account for minors, options, application domains, etc. To know where we are in the curriculum we have the concept of curriculum years and terms. This can be something like freshmen, sophomore, first quarter, or second semester. Using this convention we know that specific courses are supposed to be taken a specific times in the year, for example your second semester of your sophomore year. We then have the concept of placeholders (aka requirements) in a curriculum. Curriculums also are associated with a department (aka program). This, for instance, would keep track that the Computer Science minor would come from the Computer Science department. Placeholders have a regular expression that matches course numbers in the Courses table. This determines what course(s) can fulfill that place holder. Placeholders can also have labels. This determines what color the requirement will be in the flowchart and worksheet.

Finally we have the concept of user access. The system changes level of access and functionality based on what role the current user has. Each role then has a set of privileges (privs). This determines what functionality that role has access to. New roles can be created during the life of the system and can have different privs assigned to it.

**UI Rationale**

We have a requirement of having a ‘flowchart view’ in our system. We initially looked a just making a drag and drop version of the current SE flowchart. After discussions with the customer we felt that this didn’t add any value to the system. A mock up of the flowchart is below.
On the top of the flowchart we have listed AcaYears because we are showing actual points in time. The courses in the completed section are added by a course import subsystem. These classes cannot be edited because they are history, and you cannot change history. The color blocks below the classes are the requirements (placeholders) that those classes are fulfilling. Multiple requirements may be placed onto a single class for double dipping of classes. The Year buckets on the bottom are the requirements that need to be fulfilled in that Curriculum year. These buckets are filled based on the curriculums that are currently assigned to the student. The user is able to drag and drop the requirements from the buckets to the planned section of the flowchart. Once the requirements are in the planned section they can be modified to actually type in a course number to indicate they are specifically taking a course.

We decided to go with this design because this gave the customers the opportunity to change the flowchart and compensate for faults that the current one has. For example if the first couple terms of a student’s record does not follow the flowchart, the flowchart becomes pretty much useless as a planning tool since all of the suggested terms to take classes has been changed.

**Process and Product Metrics**

We decided to use overall time spent, time spend per issue, feature completeness, and estimation accuracy. Overall time spent entails the entire time put towards the project. Including but not limited to designing the system and documenting decisions made during meeting. This measurement is very important because it ensures that each team member is contributing an equal amount of time to the project. It is also important because this time is a required measurement by the software engineering department. At the time of writing this document 786 man hours have been put into the project with an additional 10-20 expected to be put in by the time of product delivery.

Time spent per issue is a more granular measurement than the overall time spent. Time is logged to individual “issues” which have specific types. The time is important because it helped us create more granular tasks. On average we spent 3.8 hours on a task.

Feature completeness is a metric that we used to track our progress in implementing features of the project. Features are broken down into a very granular task list. This list is then broken up and assigned to each team member. As members complete these small tasks the percentage of Feature Complete increases. The time from each of the tasks can also be combined and a rough estimate of completion of the feature based purely on time can be derived.

We used estimation accuracy to better predict our schedules and coding time for each iteration. Since the time spent on each issue is monitored and at creation the time to complete the issue is estimated. We can easily derive the estimation accuracy. Estimation accuracy is an important metric because as time goes on we should be able to gauge the amount of work needed to create a task. Studying our estimations vs. our actual allowed us to make more accurate future estimations. Our average deviation was 24.36% under estimated throughout the project with our greatest over-estimation at 100% and our greatest under-estimation at 1450%. This means that even though we tried to adjust for our error in our estimations, it wasn’t enough. This may have spawned from a lack of making our tasks granular enough. Many tasks were “Implement X feature” which can be very large. We believed in the beginning that the features would be easy to implement because our knowledge of the domain and technology was increasing but so was the
difficulty of the tasks. Our failure to realize this caused our estimations to be inaccurate during the project.

**Product State at Time of Delivery**

Overall all of the core features of the product have been implemented to a usable degree. Our system has a flowchart view and a worksheet view. While these do not mimic the worksheet and flowcharts that the academic advisors and students use they do provide the functionality that was described to us by the customer. This just comes down to a presentation and styling issue. This was a common problem during our iterations. We would commit to quite a few features to implement which would leave little time to usability design and refinement.

Our system also supports curriculum generation. Curriculums can be made with multiple terms and multiple years with a mix of semesters and quarters. Duplication of curriculums is also possible on our system from the curriculum screen. This works except if the curriculum has annotations then the annotations are not copied. The customer expressed that they wanted to choice to either copy all or none of the annotations when they copy a curriculum. This feature was not implemented because it came up during the last few weeks of the project and was set aside as there was more important tasks to work on.

Since we needed test data to operate on the first feature we developed was the student record importer. We have also written an interface so other import implementations may be written in case the incoming dataset changes. As with other features the usability of the feature is not complete. During an import, especially during a long one, there is not feedback to the user as to the progress of the import. When looking at a list of the students currently in the system it takes a very long time for the page to load. The load takes so long that the connection will sometimes time out. We talked about implementing pagination but were unable to get to it because we ran out of time.

Other features like saving a students worksheet, assigning curriculums, creating annotations, log in via RIT LDAP, and scheduling planning have all been implemented but their usability has not been tested. They have also not been tested all in concert with each other with a complete dataset with multiple users all executing at once. The enforcement of user roles and access has also not been tested. We have designed our system to handle this but have not tested it. We have included roles into our database design but haven't had enough times to properly create and test an admin screen for user roles. A strange error we would get during development was Tomcat would give us a 'PermGenSpace' error. This resulted because Tomcat has run out of PermGen space. Our solution was to simply increase the PermGen space in our Tomcat config. While this isn't a permanent solution it did decrease the number of times it happened. The problem would usually spawn after a fresh deploy of the system.

**Project Reflection**

Looking back at the project we are proud of how we worked together and the state of the project at time of delivery. Even when we felt like we were behind in development according to our schedule we were always on track with feature development.
All of the team members were able to work really well together. At the beginning of the project we considered each other friends and still do even at the end which says a lot about how we worked together. Some of the team members had personal issues that they needed to take care of and everyone was understanding and was able to deal with it. Our team was also very knowledgeable in terms of technologies, coding, tools, and software engineering experience.

We were also able to use a great set of tools. Using Redmine as a issue tracker really helped us keep track of our metrics and our progress with the product. It also gave us the ability to share our documents and status with the customer. Having this website gave us 100% transparency with the customer so they could identify anything they felt unhappy with. We also used Hudson for continuous integration. This allowed all team members to know exactly when the system has become unstable and it allows for all our tests to be run on every push to our Mercurial repository.

While there were many good parts to our project it wasn't without its faults. While having many tools assist us it also takes time to maintain them. In the beginning we decided to create development VMs so each team member didn't have to create their own environment. This also removed variables when trying to debug odd bugs. The down side was it took quite a bit of time setting up the VMs which left the developers doing nothing but documentation and prototypes to get used to the technologies we decided on. We also would forget to log our time into Redmine for our tasks. This meant that our progress was typically inaccurate for each task.

During the initial part of the project we went through extensive requirements elicitation. While normally this would be a good thing, it wasn't for us. We were initially put into a requirements analysis paralysis because our chosen technologies were not dictated to us until week 4 of the project. So the only thing we really could do was go over the requirements with the customer.

Another problem we had at some points during the project was a lack of communication. At some points team members would disappear from communication or we would update each other on our statuses when asked. It didn't help that we didn't regularly update our Redmine issues as well.

The last problem we had was customer feedback. The entire project we have had nothing but positive customer feedback but that in of itself was a problem. Since the customer couldn't identify anything that we were doing wrong then we didn't what we should improve upon. While we were good at finding problems ourselves it would have been nice to have some direction from the customer as to what they thought about the product.

Definitions

Curriculum - Generic flowchart. This can be an undefined number of courses and there pre-reqs strung together over multiple terms. Example: The courses required to complete a minor is a Curriculum

Divine Curriculum - A collection of Curricula required to graduate.

Populated Flowchart - A curriculum with courses that have already been taken and populated (This information comes from the Registrar and is the official transcript). AKA "The Source of Truth"
**Planned Flowchart** - An unmodifiable Populated Flowchart that a student has added classes to be taken in the future.

**Blessed Flowchart** - A Planned Flowchart that has been approved by an Academic Advisor. This means that all planned courses are correct on the flowchart.

**Requirement** - A specific place in a curriculum that needs to be fulfilled by a course. Example: The SE Process Elective is a requirement because it can be fulfilled by several different courses. Example: SE 362 is a requirement because it is a singular course that you have to take.

**Gospel Transcript** - A transcript that has come from RIT's registrar.

**Course** - The course in the college that has a title, course number, description and is offered by the university. Example: SE 4010-361 is the course 'Software Engineering'. It has a course number, pre-reqs, and a description.

**Class** – (AcaClass) An instance of a course. Has a section number and can be taken by students. Example: SE 4010-361-01 is the course 'Software Engineering' that is taught by Prof. XXX, meets in room 70-XYZ, and occurs at 2-4pm.

**Academic Term** - A grouping of classes that a student takes during a set amount of time during the school year. Example: A quarter is a term and a semester is a term.

**Academic Year** - A grouping of 2 or more Terms

**References**


2. [http://struts.apache.org/2.x/docs/big-picture.html](http://struts.apache.org/2.x/docs/big-picture.html)