O-PASS

(Online Program Advising and Scheduling System)

Team Flowriders

Peter Janak	Travis Lazar
Eric Herman	Christopher Johnson

Project Sponsors

Jim Vallino, Lana Verschage, Sarah Mittiga, Ed Hensel

Faculty Coach

Daniel Krutz

Project Overview

The Online Flowchart Advising and Scheduling System is designed to move the current paper only advising system to an online system. The system is split into two subsystems: the advising portion and the administration portion.

The administration portion allows department administrators to create the course curriculum for a particular year and permits the administrator to generate reports for course scheduling. The curricula will be versionable so that they can carry over from year to year or be changed. The system will also provide a high degree of configurability to define courses, programs, and display formats.

The advising portion will allow students and advising staff to create and experiment with different planned flowcharts. These plans can be saved for later use. While the advising staff is working with students, they will be able to view their plans in both flowchart and worksheet form. The administrative system can use these saved plans as a data source to predict how many students are anticipated to take a given course.

There are several different groups of end-users for this project. The first is department administration. They will build the curriculum, that all students and advisers must adhere to for the given academic year. The second are the academic advisers. These advisers have access to all of the student data and assist the students with reviewing their academic careers and planning their future. The advisers also have the ability to "bless" a student planned curriculum. The last group of users is the students enrolled in the program. The students will be able to fill out the curriculum that the administration created and plan for future terms. These plans can be shown to and approved by the academic advisers.

The project initially spanned over the Winter and Spring quarters of the 2010-2011 RIT academic year. That resulted in approximately 22 weeks to complete the project, deliver the product and all supporting documentation, and give 1 mid-project presentation and 1 end-of-session presentation. All previous development for the project was completed by week 8 of the 2010-2011 Spring quarter. Development of the product was done by the senior project team consisting of Andrew Bona, Sean Madden, Bryan Schick, Josh Peterson, and Jeff Kelley. They were in charge of development, design, and documentation of the product. The senior project sponsors Lana Verschage and Dr. Ed Hensel provided the primary feedback as to features, development, direction, and schedule. Sarah Mittiga and Dr. James Vallino from the SE department also provided feedback as the project progresses.

The project continued to be worked on during the Winter and Spring quarters of 2011-2012. This added another 22 weeks (25 weeks including Winter break) to add on to and improve upon the project, deliver the updated software, modify the supporting document, and give a mid-project presentation and an end-of-session presentation. All new development on the project will be completed by the end of the 2011-2012 Spring quarter. Continued development has been done by the senior project team consisting of Peter Janak, Chris Johnson, Travis Lazar, and Eric Herman. They are in charge of development, design, and documentation of the product. The senior project sponsors Lana Verschage, Dr. Ed Hensel, Sarah Mittiga, and Dr. James Vallino have been providing us feedback as to features, development, direction, and schedule.

Basic Requirements

1. The system shall allow different levels of user access.
    1. Program Administrator
    2. Academic Advisor
    3. IT Administrator

4. Student

2. The system shall display student program record in flowchart, worksheet, and IAP formats.
    1. The flowchart is a visual display of the course curriculum displayed as a grid.
    2. The worksheet is a spreadsheet of all required courses for the student with empty information about quarter taken,
        course number and grade.
3. The system shall allow advisors to be able to manually add, delete, and modify student records and grades.
    1. The system shall alert the user to course prerequisites which are not met
    2. The system shall keep track of credit count
    3. The system shall check against ABET and NYS credit counts
4. The system shall be able to transfer credits from quarter based courses to a semester based system.
5. The system shall allow advisors to be able to import and export student records.
6. The system shall show completed and planned courses in the student program record. There shall be a distinction between
    the completed and planned courses.
7. The system shall display the number of students planning to take a course in a given semester/quarter.
8. The system shall allow Program Administrators to input a curriculum for a set of students.
    1. Major curriculum
    2. Minor curriculum
    3. Manually or through import
    4. The system shall allow the user to duplicate a particular curriculum for other students
9. The system shall allow the user to create a note for student records and courses.
10. The system shall allow only one user to modify a particular student's program record at a time.
11. The system shall maintain an audit trail of user edits to records, including the editing user, the time of the edit, and
    the record edited.
12. The system shall allow users to access it over the Internet.
13. The system shall allow interoperability with different types of databases.
    1. MySQL
    2. Oracle
14. The system shall have disaster recovery backup and restore support.

Constraints

The application must be able to interface with an Oracle Database and a MySQL Database, with Apache, Tomcat, and Struts 2. JQuery will also be used in order to enhance the user experience.

Further Constraints:

- Web-based for universal access

- Generic enough for any department at RIT
- Supportable by ITS
- Deployable on ITS system
- Artifacts must be acceptable by ITS for development

There are a few important software quality attributes for this project. These include adaptability, correctness, flexibility, and maintainability. First we will talk about correctness. This system has to be accurate, otherwise course planning will be off, which will cause issues for students come graduation time. To that end, we will require 100% parity with SIS. That is all course information must be correctly linked to students, including grades and when the course was taken.

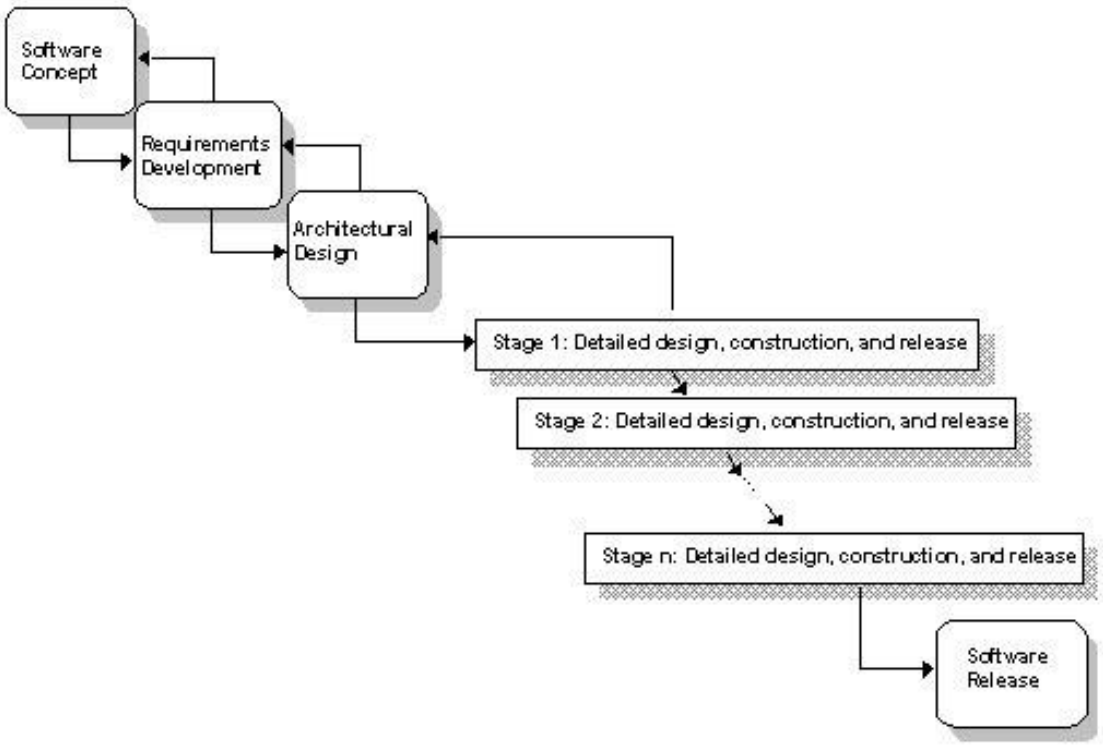
As far as maintainability goes, this system will likely be modified in the future. So far there have been two separate teams working on this project one after another. This was necessary because more features needed to be added and old features needed to be polished. We anticipate the need to continue building on this system in the future, so it will have to be maintainable. So, a new team should have the ability to easily add new features to the system and modify old ones. A system administrator should also be able to easily update the system.

Adaptability and flexibility go hand in hand here. The system will need to be able to adapt to new courses and possibly new scheduling concerns in the future. One very important part of the system is customizability. The flowchart, worksheet, and IAP view will likely need to be customized to some extent for some students. The system, therefore, should be have a good common baseline which can be built upon if necessary.

Development Process

Our team decided to use Incremental Development (aka Staged Delivery) as the process for developing the system. Incremental Development utilized a series of well-established time-boxed phases where business value is delivered in cross-discipline iterations. Incremental and iterative Incremental is defined with the following six phases:

1. Software Concept
2. Requirements
3. Architectural Design
4. Detailed Design
5. Construction and Testing
6. Release



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 also allowed us to get early warnings of problems. We planned to deliver releases early and often, and we received frequent, indisputable progress reports back from our customers. Either the release was done on time or it was not. The increments work quality is obvious from the release's quality. If the development team seemed to be in trouble, then we would be able to assess the issue within one of the first increments rather than when 90% of the project was completed.

Many Senior Project teams have utilized agile processes during the course of their projects, we felt that the nature of this project does not have the requirements flux that would warrant such a process. The customer has a good idea of what they would like to see delivered and we felt that they have communicated this to us in an effective manner. The remaining variability in the project involved certain non-functional requirements such as the technology to be used as a core architecture.

Additionally, the structure of incremental development allowed 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 would not have provided the requisite visibility, and that we did

not have the experience to make an agile process work. Iterative was a natural choice for this project.

Project Schedule: Planned and Actual

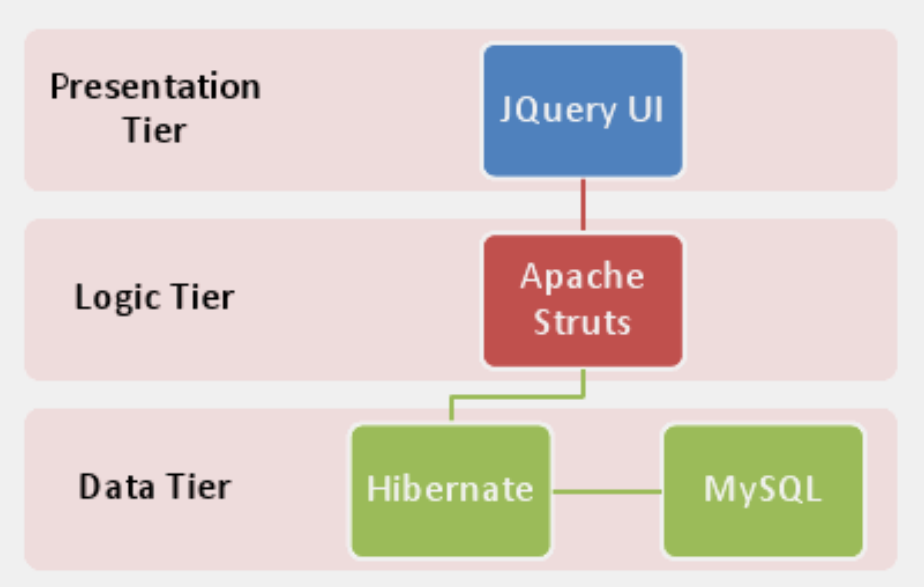
Our project schedule is reflected by the Iterative Process that we used to develop the system. Within the process there were four major scheduled releases at the end of each stage in development.

Project Release	Completion Date
Release 1	02/10/2012
Release 2	03/09/2012
Release 3	04/06/2012
Release 4	05/04/2012

The actual schedule was relatively close to what was planned. The first three releases were available upon the completion dates estimated at the beginning of the project. The final release was pushed back some due to necessary changes and bug fixes in the system. That is why in the initial planning, we left ourselves three extra weeks, up until graduation, to handle these issues before we were able to get a stable final release. Now, looking back at our original estimates, we believe that this was appropriate, and giving ourselves the extra few weeks has allowed us to get the system to a point in which the customers and we are satisfied.

System Design

We split our system into three distinct tiers: the data tier, the logic tier, and the presentation tier. Each tier can communicate directly with the tier below it and above it. We felt this separation of concerns would make it relatively simple to interact with the system and possibly extend it in the future. Most of the architectural decisions were holdovers from the prior team. We chose to stick with the technologies already in place (with exception of the database technology) and use it to form our system. Below is a further description of each tier.

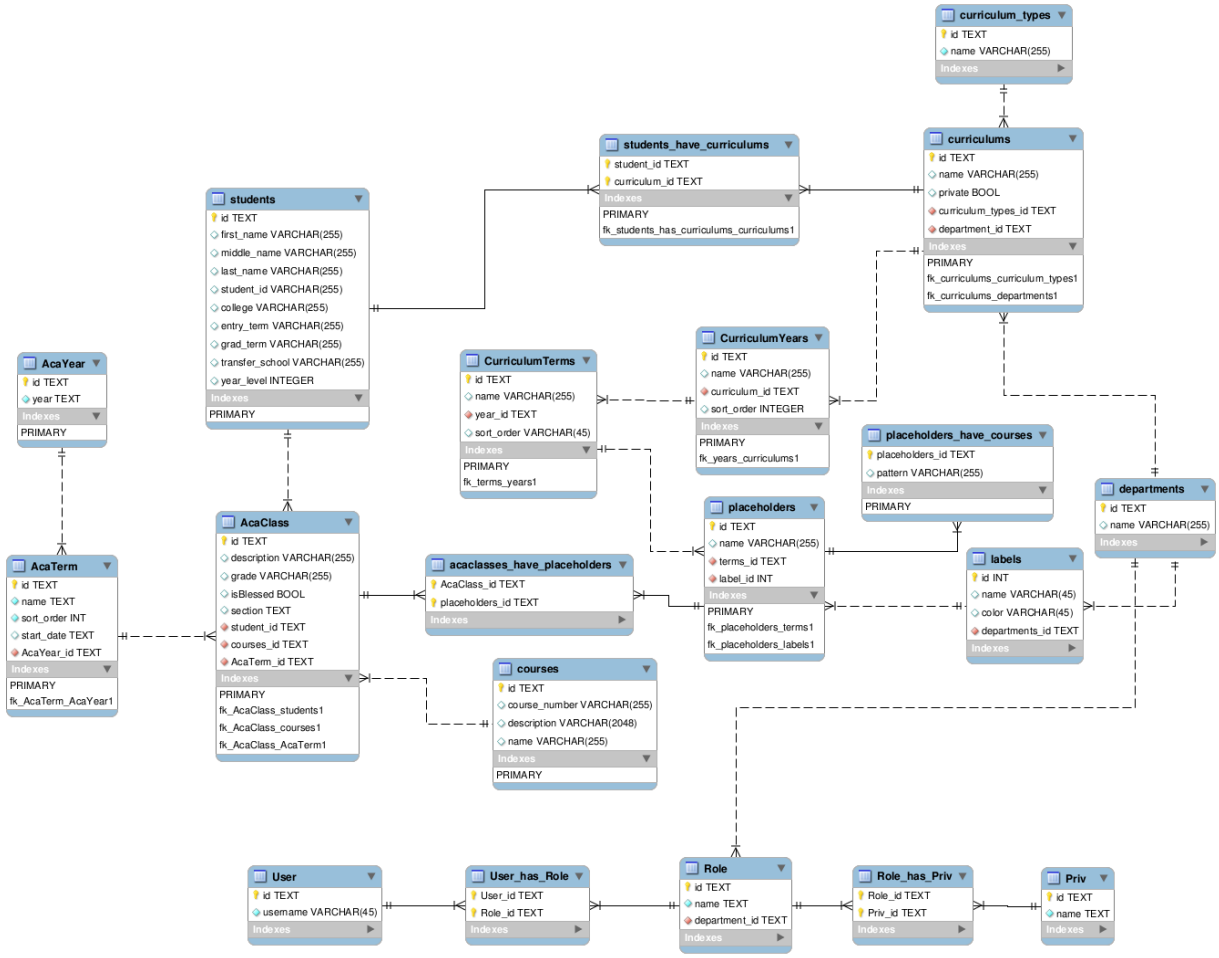


Data Tier

The data tier is comprised of a MySQL database as well as Java classes which map to specific tables using Hibernate. Each table in our database represents an entity or relationships between entities. So, for example, we have Students, Users, Classes. Those are all pretty standard. The most interesting table is probably StudentsPlanAndTakeCourses. The reason why this table exists is to create a way so that we could relate students to the courses they had taken, when they took them, what grades they got, and how many credits were obtained by completing the course. Another purpose of this table is to match up what placeholders are fulfilled by what courses for a given student. This allows an advisor to override the existing defaults for a course.

So, as was said each one of these tables is mapped to an entity object using Hibernate. The way that the client to the data tier (the logic tier) interacts with the database to actually retrieve and save any of these entities is via a Data Access Object. For this project we decided to create a generic DAO which could be used with any object, depending on how it was instantiated. This way we have a common interface for accessing and manipulating every object in the

system.



Logic Tier

Moving down the chain we have the logic tier. This tier was implemented using Apache Struts, which is an MVC framework for use in enterprise applications. This layer houses several Actions. Each action corresponds to a set of operations you can perform on a given entity. The action exposes itself to the presentation layer and communicates up to the data layer. So, for example, we have an Action for Students which contains all the operations one could perform on a student. We also have an action for Student Courses (which is what the aforementioned StudentsPlanAndTakeCourses table maps to) which allows the presentation layer to perform all of those actions. Each of these actions is mapped to a URL which the browser or an AJAX request can use to retrieve or store information.

It is at this level where most of the system's security comes into play. We have the system set up so that each action is privileged. That is, for every action one must have permission to perform that action in order for it to complete successfully. We manage this via a roles framework which allows assigning of a user to roles which have permissions. In this way we can create users which are students and might only have access to viewing worksheets or flowcharts, whereas advisors would be created which can also modify the worksheets or flowcharts.

Presentation Tier

The UI is composed of HTML, CSS and Javascript files. Each main UI has its own HTML, CSS and Javascript file, plus an application.css stylesheet. There are also multiple js libraries that are imported such as jQuery, jQuery UI, masonry, datatables and jEditable. For these external libraries no architecture was done, they were simply imported.

In each HTML file the structure is kept relatively simple, there is a main container in the body as well as multiple hidden forms. This is necessary for any popup you wish to have with use in the jquery UI library. On each main HTML page there is a div with display: none set and within the javascript click handlers #form.dialog is called. The library takes care of setting the visibility for you, as well as all other functions. Documentation can be found here: http://jqueryui.com/demos/dialog/

Each CSS file is self explanatory and contains no secret structure.

The javascript files each have their own special case structures dependent on the UI that you're viewing.

Within the flowchart.js file, the document.ready function does all of the generic setup of the files for you as well as calls multiple data getters to grab the appropriate data from the server. There is also a readyItems function which is very important. readyItems resets all of the drag and drop handlers on all elements in the page, this is important for when you move items and they can no longer be dragged or dropped on.

Further, the flowchart.js file treats the flowchart like a state transition system. Every drag and drop handler passes the drag and drop targets to the appropriate state transition functions who know how to translate one type of object into another. Explore the js file for explicit details.

The worksheet.js file works very much like the flowchart.js file where each object is also treated as a state transitionable object, except that in the worksheet there are far fewer transitions possible. The one major difference is the other box. There is a reserved ID 9999 that is always treated differently in that no placeholders can ever be dropped on it or exist in it. This is for courses that a student has taken that don't match within any curriculum, or haven't been mapped to a curriculum.

The curriculum.js is the simplest of them all. It uses the dhtmlx tree library which handles all of the tree displaying functionality (the free version of it). Due to the limitations of the free version we do a custom parsing of the incoming data and the "Open as Copy" functions. Parsing is done by iterating over an incoming JSON string. The data in these trees is stored in two different places (but no redundant data). The local (browser) object ID's are stored in the tree as well as the node name. In the case of courses this is the course number and in the case of everything else this is the name of the object. There is also a javascript object called realID which stores all other data about each object. It is keyed by local object ID and stores anything else that's relevant to the object in question. So for courses the tree might store {16: "name"} and the realID object would store things like realID[16].isDefault, or realID[16].genEd, etc.

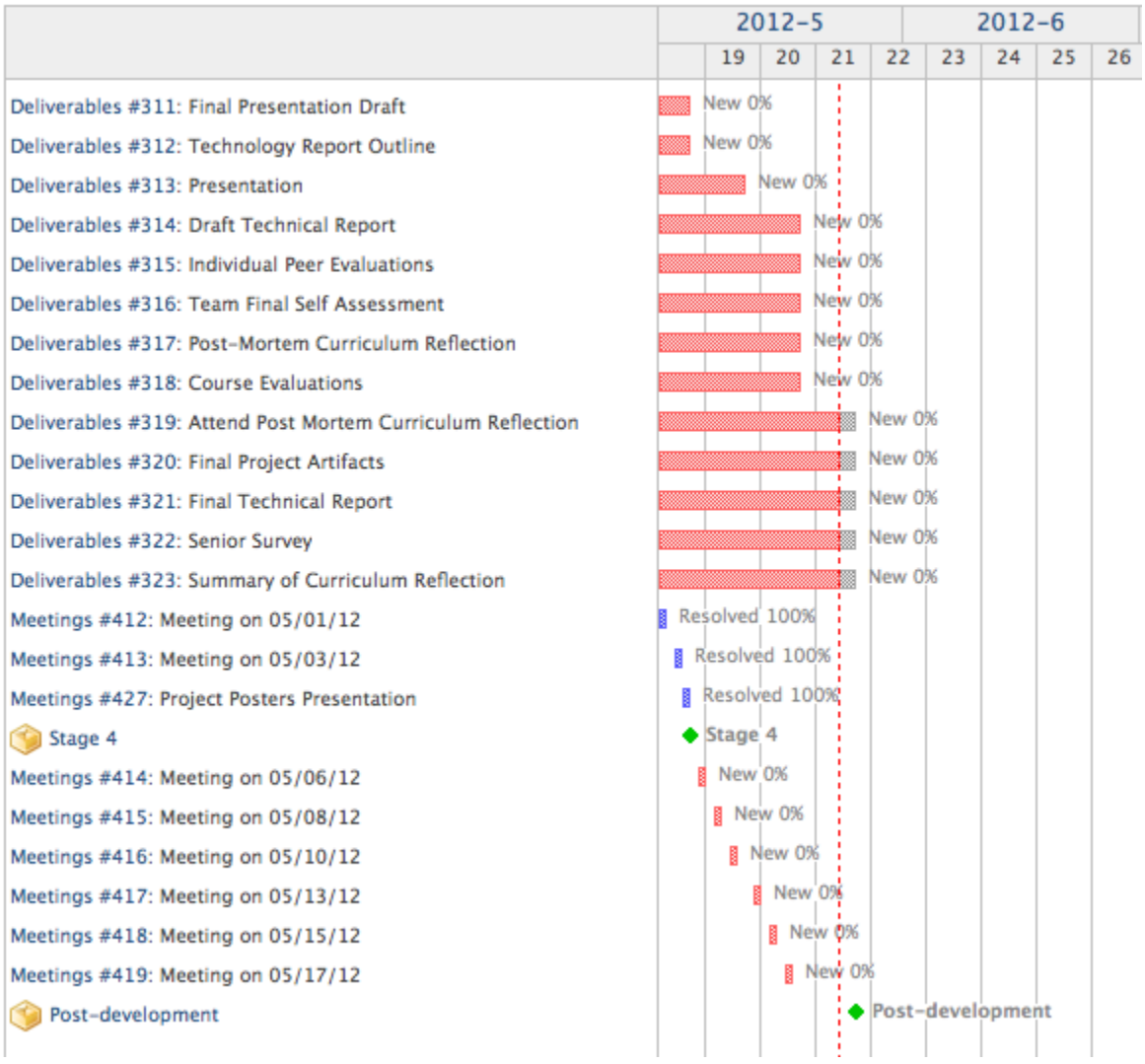
Process and Product Metrics

Redmine was used exclusively for project tracking and management. It contains a bug/issue/task tracker that have been used to report and assign anything within the project. Members of the team have updated each ticket with time measurements and any additional

information necessary during the completion of each ticket. This has allowed an extremely transparent view into the process, so that any team member or customer could view the status of the project and identify any issues. The data contained within Redmine are then used to create reports and graphs that constantly evolve and update along with project. Ideally, this has removed much of the overhead with updating documents. Additionally, these reports may be downloaded and placed into the Redmine's document area to allow for an archive of project status at particular moments such as between releases.

The exact metrics and measurements are located on the team's website: http://flowriders.se.rit.edu/projects/flowriders

Below is a snapshot of a Gantt chart that displays the final Stages of the Process:



Product State at Time of Delivery

The current state of the product contains the majority of the core functionality of the system. Many of the bugs reported have since been fixed and tested. While some of the UI is not as polished as we had hoped (the curriculum builder UI), the portions that will be used the most have been well polished and we are satisfied with them. There were no unplanned features that were added, but how some of the planned features were adapted into our system needed to be clarified and accepted by the project sponsors. There was one feature area in particular that we had negotiations with the project sponsors. This area was related to the annotations that we intended to have throughout the system. Originally, we had intended to allow users to attach annotations (text and/or files) to almost every relevant component in the system. After discussing this feature with the sponsors in more detail, we negotiated that instead of having annotations everywhere, we would implement only annotations for students and curricula. This was the only major difference between the planned features and the features that were actually implemented.

Project Reflection

What went well?

- The team had prior knowledge of the technologies being used, so this allowed us to spend less time learning the technologies and more time using them.
- The Iterative Development Process allowed to team to provide the sponsors with releases at the end of each phase. That way we could receive feedback and adapt further phases based on that feedback.
- The UI design looks clean, usable, and user friendly based on feedback from the sponsors.
- JUnit Testing for the back-end data of the system.
- Manual User testing for the front-end UI based on state transitions.
- Negotiations with the sponsor about the priority of each feature and how we narrowed the scope of the annotations of the system.

What went poorly?

- Merging of initial documentation caused issues and discrepancies that confused the project sponsors.
- Equally distributing project tasks was difficult based on the number of tasks in each phase and who specialized in which technologies being used.
- Documenting exact time measurements in Redmine was a challenge. It was easy to forget to start and stop the timer when in meetings or working on tickets, so the time measurements are not exact.

In the future, I think the only aspect that we would do differently is the documentation portions of the project. We are satisfied with how the implementation of the system turned out, but because we focused heavily on the development, our tracking of time and updates to documents took a back-seat. We still followed our process pretty closely with the exception of the delay in release 4, but the metrics aspects should be more precise in the future.

A lot of the things that our team learned through this project are related more to experience in general. It was good to be able to apply the various things that we have learned through class and co-op experiences and use them to help make this project go as smoothly as possible. It was nice to see everything put to use, and we each got a better feel for the areas we worked on in the project. This should help us with projects that we will take on in the future.

References

1. Iterative Process Diagram - Steven C McConnell - Microsoft Press, 1998
2. http://en.wikipedia.org/wiki/Iterative_and_incremental_development#Iterative.2FIncremental_Development
3. http://c2.com/cgi/wiki/wiki?HistoryOfIterative
4. http://alistair.cockburn.us/Incremental+versus+iterative+development