CARS++

Team Skyline

Corey Maher, Kevin Lakotko, Matt Bialek, Yin Poon

Project Sponsors

Dan Bogaard and Steve Zilora

Faculty Coach

Michael Lutz

Project Overview

Introduction

Most departments at the B. Thomas Golisano College of Computing and Information Sciences (GCCIS) at RIT use a software application to plan course offerings. This application is called the Course Assignment and Request System (CARS) and is capable of planning what courses are offered as well as when each course is offered, what room it is offered in, and which faculty member will teach it. Every department has a different set of procedures that is used to plan course offerings. Nevertheless, many choose to use CARS to aid this process.

Overview

As said above, CARS manages all aspects of course management. Staff can build quarters/ semesters to be offered. Faculty can request courses to teach and keep track of their current load for the year from the built quarters/semesters. Department staff may honor those requests or adjust depending on student enrollment and demand for certain classes. CARS also supports generating reports, printing schedules and course catalogs, monitoring faculty portfolios, and room utilization.

The users of CARS are ideally the entire college of GCCIS but realistically the IST (Information Sciences and Technology), SE (Software Engineering), CS (Computer Science), and NSSA (Network Security and Systems Administration) departments of GCCIS as well as the Dean's office. The users are familiar with previous versions of CARS and look forward to new features that will shape the future of CARS.

Goals and Scope

The scope of this project is to re-implement the current system, update, where necessary, to newer technologies, and implement missing functionality. One major change to the current system is to provide the functionality to accommodate all standards in the new semester system. A problem that needs to be solved is the ability to deliver data to additional platforms with a possible service oriented architecture. After finding security holes in the existing system, security also becomes an important aspect of the system which made be taken into account.

Out of scope of the project is the ability to integrate with GeneSIS, and automatic addition of

quarters/semesters added to SIS.

Basic Requirements

The basic features of the system can be broken up into nine different categories: User Authentication and Authorization, Course Selection, Course Assignment, Reports Generation, Account Summary, Terms Management, Courses Management, Rooms Management, and Users Management.

User Authentication and Authorization

Users of the system will need to login to and be authenticated by the system. The system will also be able to identify what capabilities each user should have, and provide functionality accordingly. Unauthorized access of the system shall not be permitted.

Stimulus/Response Sequences

1. The CARS prompts for account username and password.
2. The user enters username and password.
3. The CARS validates the combination of username and password.
4. The user is authenticated and logged in if the combination is valid.

Course Selection

Faculty users of the system shall be able to select courses they wish to teach in the open term. The system includes courses from multiple departments for selection. Upon selection of courses, the system shall be able to generate a schedule with the courses selected accordingly.

Stimulus/Response Sequences

1. The CARS displays a list of courses ready to be selected.
2. The faculty user selects a course by dragging the course and dropping it into a course box.
3. The CARS displays a feedback about the course is successfully selected for the specific term.
4. The faculty user selects the “Show Schedule” link to generate a temporary schedule of the selected courses.
5. The CARS displays the schedule.

Course Assignment

The staff and chairman users shall be able to manually assign an instructor to a specific course with specific amount of credit. The system includes courses from five departments for review, which includes Computer Science, Interactive Games, Information Science, Network and Security, and Software Engineering.

Stimulus/Response Sequences

1. The CARS displays a list of courses ready to be reviewed.
2. The user selects a course on the list.

3. The CARS displays information upon the selection.
4. The user selects a specific instructor and credit amount, and assigns them to the selected course.
5. The CARS adds the instructor with credit amount to the assigned faculty list.

Reports Generation

The system shall be able to generate different kinds of reports. Users of the system shall specify the report type, scope and term to generate a particular report.

Stimulus/Response Sequences

1. The CARS displays a list of report type.
2. The user selects a report type.
3. The CARS displays a list of scope for selection.
4. The user selects a scope and term for the report.
5. The user selects “Show Report”.
6. The CARS displays the particular report.

Account Summary

The system shall be able to generate a course schedule of the assigned and requested courses of the user on a specific term.

Stimulus/Response Sequences

1. The user selects a term for the assigned or requested courses.
2. The CARS generates and displays the course schedule with the assigned courses or requested courses for the specific term.

Terms Management

Admin users of the system shall be able to create and delete a term. They shall be able to modify the status of a term or do a rollover of courses from one term to another. The system shall perform these tasks only to the selected department.

Stimulus/Response Sequences

1. The admin user enters a term number in the “Create Empty Term” textbox and selects “Create Term”.
2. The system responds a feedback about successfully created the new term.

Courses Management

Admin users of the system shall be able to create new course, section and section meetings for the course. The users shall be able to modify the course and its section details. The users shall also be able to delete a section if it has no requests or assignments. They shall also be able to delete a course if it has no sections.

Stimulus/Response Sequences

1. The CARS displays a list of courses of the selected department and term.

2. The admin user selects a course from the list.
3. The CARS displays information of the selected course.
4. The admin user makes modification of the information and selects “Submit”.
5. The CARS displays a feedback about successfully updated the course information.
6. The admin user selects the close button to return to course management list.

Rooms Management

Admin users of the system shall be able to add and remove room on the system. Information of any room can be edited and updated at any time.

Stimulus/Response Sequences

1. The CARS displays a list of rooms of a selected building.
2. The admin user selects a room.
3. The CARS displays information about the room.
4. The admin user updates the information by selecting the “Update” button.
5. The CARS displays a feedback about successfully updated the room information.

Users Management

Admin users of the system shall be able to manage other users of the system. Functionalities include add a new user, modify and update information of a user, remove a user on the system and set permissions of the user.

Stimulus/Response Sequences

1. The CARS displays a list of users of a selected department.
2. The admin user selects a user on the list.
3. The CARS displays information about the user.
4. The admin updates the information by selecting the “Update” button.
5. The CARS displays a feedback about successfully updated the user’s information.

Constraints

Time

This project started in December last year and its release date was set in May 2012; therefore, the team had about five months to complete the project.

Personnel

In terms of human resources, the team comprised four R.I.T. full time students who had workloads from other courses besides this project, unlike regular full-time staff. Manpower was expected to drop in examination period (week) and in holiday period. Each team member was expected to carry out an average of 3 hours of manpower every day.

Money & Other Resources

This project did not have any funding because this project was mainly for educational purpose. Therefore, choices of applications or software that we used to aid our development had to be free

or available to the Software Engineering department.

Technologies

There were technical constraints that were taken into account in the development of CARS++. Since the system is for the Information Science and Technologies department servers; it needs to be compatible with that environment. CARS++ is web based, accessible through a web browser and it supports Internet Explorer 7+, Firefox 3+, Chrome 14+ and Safari 5+.

Server Technology Constraints

Since this system will be running on RIT server hardware, it must follow the RIT school standards. As described in the introduction section, one of the major change of this system from the old system is to provide the functionality to accommodate all standards in the new semester school system. Therefore, this system was designed and built according to the semester standards.

Development Process

Project Lifecycle

Early on in the project the team determined what aspects of a project lifecycle they thought would be beneficial for this project. The final list of aspects that were decided upon where: risk centric, high visibility, and incremental.

The lifecycle that best addressed these aspects was the spiral lifecycle. This lifecycle was chosen over others due to its ability of addressing risks early because the risk analysis happened at each iteration. This lifecycle also allowed greater visibility to the sponsors by having a potentially releaseable product at the end of each iteration.

The team then outlined the process methodology to the sponsors and got approval to move forward with it.

Project Organization

Roles and Responsibilities

Due to the small size of the team for the project, roles were combined to form one role per member. On top of these major roles each member was also considered a development engineer. This allowed each major aspect of the project to be accounted for. A person filling a role that was short on time and could not complete a responsibility at that time would be responsible for distributing the work among the rest of the team. The leads were held accountable for their aspects of the project whether they completed them or not.

Team Lead

The team lead is responsible for the successful completion of the project defined by the project sponsors and faculty coach. This involves defining and maintaining overall project vision, scope, direction, and constraints which is then distributed to the other leads.

Kevin filled this role.

Technical Lead

The technical lead is responsible for the technical decisions regarding the project. The technical decisions include design, and architecture of the product. They will also be in charge of construction, integration, product builds, development environments, and deployment.

Matt filled this role.

Requirements and Planning Lead

The requirements and planning leads is responsible for all requirements related tasks. These tasks include gathering, maintaining, and tracking requirements.

Yin filled this role.

Quality Lead

The quality lead is responsible for the testing activities within the project and ensuring that the project goals are achieved. These activities include creating test plans, creating a process for code reviews, and maintaining the automated testing infrastructure.

Corey filled this role.

Project Schedule: Planned and Actual

We used the project timeline we found on myCourses as a base when we planned our project schedule. The project timeline contained all the deliverables and their due dates that we needed to submit to the SE department for the project. We estimated project planning, design and documentation would take about a quarter to complete. Therefore, we planned to complete all pre-development work in the first term and work on implementation in the second term.

Since we decided to take a bottom-up implementation approach, we first made our estimation on the time that data and business layers would take to complete, then the presentation layer. We noticed the business layer could get complicated due to complex requirements because all departments seemed to have a different set of procedures in planning an upcoming term. Therefore, we figured business layer implementation would take up about the first half of the second term and the presentation layer would take up the remaining time. Based on our plan, we had set the following milestones.

First Term:

- Release 1 before end of the term

Second Term:

- Release 2 with some back-end implementation by end of week 2
- Release 3 with back-end completed and some front-end implementation by end of week 5
- Feature Freeze Release by end of week 8
- Final Release with all front and back end implementation by the end of May (end of term).

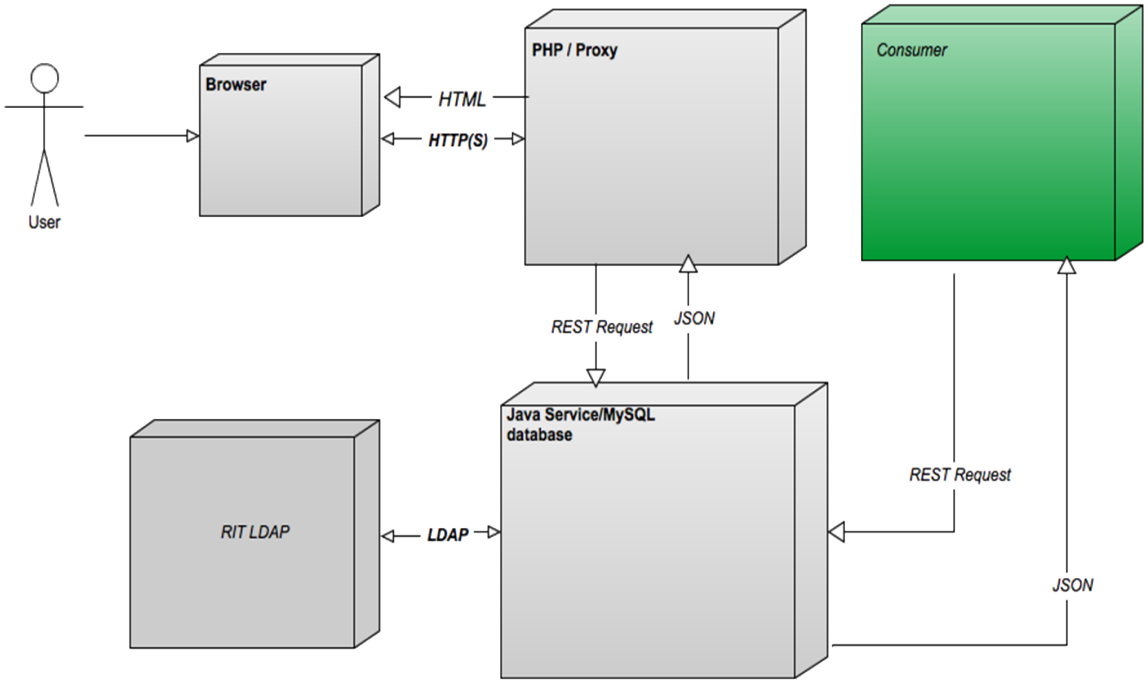
In the actual outcome, we could not follow our planned schedule exactly but we were able to use it as a reference. We were able to follow our schedule up to week 2 in second term when we realized we could run out of time on front-end implementation. Therefore we selected two of our teammates to begin working on the front-end rather follow our original plan. In week 5, we were

not able to finish all back-end implementation as planned because we were developing with two less developers since week 2. But fortunately with all the hours we have put in, we managed to complete all requested features by the end of week 8 and handed the project to the sponsors at the end of term.

System Design

Our system architecture follows a three tier model. On the front end we have our web site which is the main interaction for the user. In this level the technologies used are: CSS, HTML, jQuery, and Twitter Bootstrap. The next level is the PHP server which serves up HTML to the users browser and also makes the calls to the java service to retrieve the needed information. In this level the technologies used were PHP, and CodeIgniter, a PHP framework. The third level is the java service which handles all data within the system. This server is called through the usage of our rest API. When a call is received the server processes the necessary logic and responds with the desired data in JSON form. This service is implemented in java with the Play! framework with Hibernate handling the database interactions. Through the usage of this service a third party system could call our API. This is known as being a consumer of our system. To be able to complete the call the consumer needs to be a registered consumer of the system through the generation of a consumer token. This consumer token must be used for every call from the system. On top of this any call to the back end also needs a user token. This authentication is done through the means of RIT LDAP authentication. Pre-determined users can have backup passwords or user names which are not related the RIT LDAP. This design was created from discussions with the sponsors, which we agreed with. The reason for the heavy involvement in the process was because it would be hosted on their servers. The technologies and servers we used had to work for them because in the end they would be maintaining it.

The Java service is also comprised of three layers. In the first layer we check if the user is authenticated through RIT LDAP. Once they are authenticated they are able to make calls based upon their role in the system. These roles include Super Admin, Faculty, Administration (Staff), and Department Head. Each role has specific permissions. When the call is made on behalf of the user the call goes to a certain controller that handles each model. Each model has a controller that is specific to its actions such as create, update, delete, read, and a few resource specific methods. When the URI hits the back-end it is delegated to the specific controller which will handle the business logic of action. Each resource is defined as a model which has specific business rules to itself such as the name must be no greater than 255 characters. If the action requires interaction with the database, the business logic interacts with hibernate to carry out the action. Once the action is completed a response is returned to the caller in the form of a status code or JSON response and a status code. This is the basic interaction of most calls to the database.



The graphic above is the deployment diagram of this system. As mentioned earlier, the CARS++ system consists of two servers, the Java service which contains all of the business logic and data for the system, and the PHP server which creates the views by making requests to the rest API. The CARS++ system also interacts with the RIT LDAP server which is out of our control. The green "Consumer" box in this diagram is a theoretical client that is interfacing with the CARS++ system through the rest API. It was included to show that there is not anything inherently special about the consumer, or PHP server, that we are providing with CARS++. Any other developer can create a system which interacts with the java service through the rest API, provided they are given access. The final aspect of this diagram is the Browser and User. As mentioned earlier the PHP server generates HTML views of the CARS++ system to users through a browser, which is outside of our control. The CARS++ system does not include a browser, or require any modifications to a user's browser.

Alternate Design

Another design that was considered during the planning stage was having only one box which would house all the needs of the system. In this variation the PHP server, java service, and database would all be on the same box. This was considered but was not the ideal situation for both scaling and security. This was a bad decision because we wanted to have the PHP server acting in the middle to serve up the web pages to the browser and also to forward the calls to the back-end.

Process and Product Metrics

The set of metrics that were tracked are split up into several categories: progress, defects, and

tests. We used Redmine to assist us with tracking schedule and effort metrics. The following table describes each of the metrics we planned to use and the rationale behind our choices.

Name	Category	Rationale
Slippage Chart	Progress	The slippage chart is used to display how far a project is off schedule. It is used to show how many days the project is ahead or behind the schedule, based on planned dates for milestones. Since the project is time-boxed to a relatively short period, any slippage in the schedule can doom the project. Therefore, when any slippage occurs it must be caught early in order to bring the project back in line with the schedule. Normally, this metric would require extra overhead to plan on specific dates for reaching milestones, but since the project will be following the spiral model milestones will already be planned. Therefore, this metric can be tracked without too much extra overhead.
Defects by Type	Defect	Defects by type will display how many defects the project has for each given type that is defined. It is used to show the current state of defects in the system and can be used to determine where more focus may need to be shifted. This metric can also be taken over time to see how the defects by type change from week to week or milestone to milestone.
Defects by Priority	Defect	Defects by priority will display how many defects the project has in each given priority level. As with the defects by type it will give another view of the current state of the defects in the system and can be used to determine if focus must be given to specific defects. This metric can also be taken over time to see how defects by priority change over time.
Test Cases per Requirement	Test	Test cases per requirement measures the number of test cases that are associated with a single requirement. This metric can be used to give a quick overview of each requirement and ensure that each requirement has at least some tests verifying that the system fulfills it.
Code Coverage	Test	Code coverage is a metric which measures the degree

		to which the source code of the system has been tested. This metric is easy to measure and gives a general indication on how well the tests run through the different code paths in the system.

Results

Slippage Chart

Unfortunately the overhead of adding each feature to be developed into Redmine became too great early on in the project. We were therefore unable to track granular slippage in individual features that we could aggregate into an overall slippage for the project. We instead relied on the schedule to determine what deadlines we had to make. This did make determining whether the team was on track to meet their deadlines more difficult within an iteration. Fortunately, due to the size of the project, and skill of the team, all deadlines were made.

If we were to continue this project we would use Redmine to its full potential to keep track of the slippage of features.

Defects by Type / Defects by Priority

As mentioned above, unfortunately the overhead of adding defects into Redmine became too great for the team. We were therefore unable to track defects in the system and do not have any metrics in this category.

If we were to continue this project we would use Redmine to keep track of defects so we would have meaningful metrics in this category.

Test Cases per Requirement

In order to gather the test cases per requirement metric we had to follow a very manual approach of going through each functional requirement and ensuring that it was covered by at least one test case. There were several issues with this approach. Since this was a manual process it was time consuming and could therefore not be performed as often as other automated metrics, such as code coverage. Also for this metric we only cared about whether there was at least 1 test case per requirement. This unfortunately did not give us very much insight into how well each test case was tested.

If we were to continue this project we would determine how many test cases there were per requirement instead of whether there was at least one.

Although this metric was not as useful as we hoped, it still ensured that we had at least some test covering each requirement which we used several times to determine what test cases were missing.

Code Coverage

The continuous integration server used by the team was set up to automatically determine the code coverage of the tests that were run on each commit. The team was therefore able to see the overall code coverage of the project, and was able to see more specific coverage information regarding the coverage of specific files and packages, and the different branch paths that were

covered by the tests.

At the end of the project the overall code coverage was:

Category	Percentage
Classes	98%
Conditionals	66%
Files	98%
Lines	76%
Methods	83%
Packages	100%

These results show that almost every class in the system is hit by at least one test, and over 80% of the methods in the system are hit by at least one test. The areas of the system that are bringing these numbers down are the features that were partially done, but not completed due to scoping and time constraints, such as reports. Also, since the Play! Framework uses static methods for all controller methods each class will have 1 method that is not tested, the constructor, since it is never used.

While this is not the 100% code coverage that every project strives for, it is still quite high.

Product State at Time of Delivery

The product delivered at the end of our second trimester was feature complete to the standards the sponsors supplied to us in week 7. During that meeting we discussed that we might not complete all the necessary features. Together we formed a priority list. This was broken into three categories: Must have, should do, nice to do. We managed to complete all tasks besides reports. The sponsors were fine with our progress. The only feature that was planned and was not implemented was report generation. We have a base implementation of it currently in the back-end in our java service but currently do not have a web page for the feature. I would not call this 100% complete in the back-end it has not been updated or test thoroughly with these new additions to the models in the system. Currently there are some minor problems with pages which have been noted in the Known Issues document.

There were unplanned modifications for better usability. On the front end we modified the way course requests were made to make it more clear and to reduce conflicts. Also added in a easier way to handle the creation of terms. With the new term number format will be confusing for administrators, our system helps guide the user to create the proper number. When a user first logs into the system the user no longer lands on a dashboard but rather a “My Summary” page. This page displays a users requests and assignments in a calendar view.

Project Reflection

We had some things that consistently went well for our project team such as prototyping. This was especially helpful early on during the project when we had to prove our technology choices were technically sound and would fit the needs of our system. This provided a great feedback loop between us and our sponsors. We could show them the technologies we desired to use in action, and meeting requirements. Sponsor interaction in general was a great contributing factor in the success of our projects. We were in constant dialogue with sponsors through email and weekly meetings. A true asset was the fact that the sponsors were the previous developers of the system. They were able to explain requirements, and help us with some of the implicit business rules to the system. Along with talking to sponsors with requirements we held many stakeholder interviews with administration and department heads of the various departments of GCCIS. The project went well overall due to our strong dynamics and interaction with the stakeholders.

We had some aspects of our project that challenged us. Initially we had a document heavy process in our implementation of the spiral model. As time went on the weight of maintaining all documentation and tracking of time began to slip. Work was still happening but we stopped filing bugs and logging time. The tracking of bugs was moved to a google doc. We also struggled to define real spirals. Our spirals got delayed early on because we were locked in a prototyping phase to please the sponsors. This delayed us around three weeks. While this did help mitigate our technology risk there was were some struggles with technology stack at times. Getting play framework to interact the way that we wanted at times took work, and research. Defining our REST API and maintaining it was a difficult task. There was a constant struggle between what we wanted to add for functionality but for it have usage. The goal was to not bloat the API with calls that would never be used once. No one on the team had defined an API before so this was an interesting task. Deciding the URLs to maintain the rest methodology was a struggle. This took time to understand. These challenges did make senior project far more interesting than we would have liked.

If we were to do this project again in the future we would change mainly when we started to develop. As mentioned previously we were delayed three weeks due to coming to an agreement with sponsors. We planned to start development in week 8 of the first quarter but instead after. We first started focusing with everyone on back-end development. This was not the most efficient decision in that we stepped on each other's toes often and the front end was going no where. If we were to split off sooner we believe that we would have had a more complete product, with less stress on our end.

As developers we learned a great deal during this project. We learned how to handle a large six month project from project inception to hand off and all the struggles that happen in between. Estimating of tasks was a huge part because we had to make the goals we said we were going to make. Parts depended on each other so having an idea of the models ahead of time helped mitigate this risk. Along with estimation being able to plan ahead into the future for releases was a great learning experience for all on the team. This project made us stronger developers as a whole and we had a great time doing it.