Project Overview
RIT’s Office of Cooperative Education and Career Services handles full-time and co-op job searches and provides relocation information in many areas of the country. There was currently no existing system providing networked housing data for students in those areas. A need existed to build a system that allows students to find roommates and/or housing while on co-op throughout the entire country.

Such a system needed to allow students to search for roommates and/or housing based on specified preferences. The student could then view the results and request a roommate and/or house. The system would greatly assist students in finding short-term leases with other co-op students in all areas of the country.

The system also needed to provide both the capability for non-RIT users to add housing options and for administrators to monitor and turn off features of the system. The latter is important due to the uncertainty of the legality of allowing non-RIT users to post houses in an RIT-based system.

The Co-op Housing Portal currently provides all roommates and housing features. One goal of the system is for it to be released to open source. A Google Maps feature displaying the location of all results has been assigned as future work.

Basic Requirements
The primary requirements of the system were focused on allowing a student to acquire housing when they go on co-op. Second to the student functionality were administrative tasks and lastly the requirements for housing providers.

It was required that students shall be able to perform the following functions:

- Log in to the system and create a profile.
- List houses and rooms that other students can search for.
- Search and view students and houses.
- Remove or block houses from all searches.
Administrators were required to perform the following functions:

- View, manage, and maintain all data within the system.
- Enable or disable housing related features.
- Edit houses in the system.
- Edit user profiles.
- Block offensive houses or profiles.
- Search houses and users in the system.
- View, approve, and reject account requests for housing providers.
- Grant administrative privileges to other users.
- Import and export data from the system.

Housing providers were required to perform the following functions when housing provider features are enabled:

- Request a housing provider account.
- Enter houses into the system.
- View, approve, and reject student requests to move into listed houses.
- Remove or block houses from all searches.

**Constraints**

The implementation of the Co-op Housing Portal was constrained to use software that was already installed and supported on existing client systems. Additionally, the team was required to install and maintain the product on said system during the entire time the team was working on the system. The allowed technologies were the following:

- Apache
- LDAP Authentication
- MySQL
- PHP or Perl

Additionally, the team had specific requirements about configuration options that required minor changes to be made to the staging and production environments. These changes are very well documented.

**Development Process**

Originally, the sponsor had no requirements abound the development process. Team members agreed on using a modified incremental approach and began the process of designing and staging out requirements (See Figure 1 below). This was originally chosen as a method of controlling requirements changes without entirely eliminating them. We designed the two components around the major requirements in the system, breaking them apart into housing and roommate components. This allowed us to develop both pieces independent of each other. After the first quarter, the team felt confident that there was a firm grasp on the requirements and the implementation phase begun.
During the second quarter, the sponsor requested that our team change our meeting format in order to include code walk-throughs and reviews in addition to the regular demos. This helped to further address any questions about where the team was with the product and what still needed to be finished. It also helped to educate the future developers as to our development strategy and to familiarize them with the system they would be supporting in the future.

Throughout the entire process, the team continued to keep the sponsor up-to-date via the team website using Trac[1]. This helped us to be informed and updated on team members’ responsibilities and progress. Additionally, it acted as a medium to communicate and track defects. One team member was designated to be the contact point for the sponsor. This helped to prevent any miscommunications about who to contact when there were questions about the progress or overall product.

During the second quarter, we performed usability testing. This process helped us to identify key areas within our interface that needed improvement. These tests were performed only after we had an entirely usable system. These tests were led by our testing coordinator, but everyone participated in the actual testing process.

**Project Schedule: Planned and Actual**

During the first part of the project the team developed an initial project schedule. The final release dates for both quarters was set by the sponsor. From there, the team applied previous project experience to establish a first draft of the development schedule. As the
quarter progressed and the requirements phase started to slip, the schedule needed to be reevaluated. The schedule was revised between the two quarters. To ensure the team could deliver a working product by the original due dates, the scope of each iteration was slightly reduced.

The project had a few major deadlines and activities. Some of the deadlines were imposed by the sponsor (indicated by an asterisk). The remaining deadlines were decided upon by the team to ensure the project would be completed on time and meet all the requirements.

**Planned Schedule**

- Project Plan Winter Quarter, Week 3
- Requirements Winter Quarter, Week 5
- Increment 1* Winter Quarter, Week 8
- Update Project Plan Spring Quarter, Week 1
- Increment 2 Spring Quarter, Week 4
- Final Increment* Spring Quarter, Week 8

**Actual Schedule**

- Project Plan Winter Quarter, Week 3
- Requirements Winter Quarter, Week 8
- Increment 1* Winter Quarter, Week 8
- Update Project Plan Spring Quarter, Week 1
- Increment 2* Spring Quarter, Week 4
- Functionality Testing Spring Quarter, Week 5
- Usability Testing Spring Quarter, Week 7
- Final Increment* Spring Quarter, Week 8

The revision of the project plan took into account the slippage of the requirements phase and made relative adjustments to the features found in the two development increments planned for the spring quarter. Since the revision of the project plan, the only major change was the addition of functionality and usability testing. All deliverables were completed and most milestones were completed on time.

**System Design**

**Introduction**

Early on in the development process, the team had considered a variety of different approaches to designing the system. As the requirements had yet to be fully hashed out, the design evolved throughout the requirements gathering process. There were many factors that played a role in the team’s various design changes, including technology requirement changes, discovery of helpful tools, and changes in system functionality.
Throughout the development process, these factors incurred both large and small changes.

**High Level Architecture (Initial Design vs. Final Design)**

The team’s original system idea and design was drastically different than what eventually became the final design. This was in part due to confusion concerning the purpose and direction of the actual system itself. The initial system was designed such that it was to be a core back-end system providing the functionality for students to connect with housing opportunities. The system was to make use of artificial intelligence to match people and housing data based upon users’ criteria. The system would passively be searching continuously in the background, attempting to determine compatibility between users and houses, rank the found matches, and then communicate with the front-end through API calls. This was gearing up to be a backend used for front-end applications such as Facebook.

While much of the core functionality of the original design remained in our final system, the structure of it was completely changed through further requirements elicitation and discussion with the sponsor. While there were a variety of reasons for adopting the change, the key reasons were the sponsor’s concerns and desires, complexity of neural networks and other artificial intelligence algorithms, and the core technology requirements later elicited from the sponsor. After working further with the sponsor, and eliciting more requirements, the system’s design was refactored into a web application, to be housed on the Co-op and Career Center’s web space. This allowed the system to become a much clearer and defined experience for the user and a more usable and “tangible” product for the sponsor at the end of the project. Additionally, the team was able to address many of the original design’s concerns by cutting out the artificial intelligence component and utilizing the required technologies.

The team’s final system design was ultimately chosen based on a combination of factors: the sponsor’s preferred use of the programming language PHP, the frameworks available for PHP development, and general software engineering design principles the team had learned throughout their time at RIT. It was fortunate that some members of the team were familiar with PHP and existing frameworks and as such an initial, high-level design was crafted. Figure 2 displays the final high-level overview of the system.
Through the requirements gathered at this point, and the aforementioned familiarity with PHP, certain technologies were chosen that would play important parts in the system. Many of these technologies were already existing systems which were required by the sponsor to integrate with the new system. For example, users logging in and being authenticated would be handled by RIT’s DCE system, making use of LDAP. Additionally, Apache was utilized along with MySQL.

The other features of the system can be broken down into components. The three major components of the system follow the MVC design pattern: the Model, View, and Controller. The View and Web UI components consist of the actual HTML markup and PHP scripts the user is able to see and interact with. The view takes in the user’s input and passes it along to the Controller. When the business logic and processing is complete, the view is then updated and displays the new information to the user. The Controller component is then used to manage the logic of the application. The Model component represents the model classes that query and access the Database (DB), which actually stores the system’s data. The REST interface is used for building web services, which allow external applications a simple interface to interact with our system. One of the biggest advantages of utilizing the MVC approach is that it was compatible with various existing web-frameworks.
Cake PHP Framework / MVC Design

After some research, the team found an open source web framework similar to Ruby on Rails for PHP: CakePHP. As many of our team members were not familiar with PHP and web-frameworks in general, but had utilized some Ruby on Rails in the past, CakePHP was desirable as much of its setup was similar if not the same as Ruby on Rails. By utilizing this framework, much of the project structure would already be in place. This saved the team considerable effort, and allowed the team to begin working on features and functionality much sooner than if they had built the entire project structure from the ground up. Additionally, CakePHP offered a structured Model-View-Controller architecture, which was already planned to serve as the foundation of the system.

![CakePHP Request Flow](image)

Figure 3: CakePHP Request Flow

Figure 3 displays the general flow of a simple HTTP request made by a user to our system. The flow follows the general MVC pattern and request processing of most web applications. At a high level, this proved to be effective as all members of the team were familiar with web applications in some form, if not PHP directly. This lessened the learning curve and allowed all members to begin actively participating in development right from the beginning.

Shortly after the initial high-level designs were completed, the team began to work on the database table design. While creating the system design, the team made an effort to consider a basic database table structure. This proved to be helpful, as when it became time to design a database schema, much of the class structure could be “translated” directly into database tables. This helped save the team time, as it avoided redoubling the team’s efforts.

Database Design

Figure 4 shows the team’s initial hybrid class-database design in crow’s feet notation. This design came about by considering all of the features in the system and breaking
them down first into objects and then further into database tables. This served both to give the sponsor an idea of the planned system class design, as well as help the team keep in mind which features were related and disjointed. The benefit of this approach to coming up with a class structure was that it allowed the team to keep a very close relation between the individual classes and the tables in the database. Throughout the development process, our database design was refactored and many changes were made to it. The majority of the changes ended up being minor, however, which made making the changes smooth and seamless in development.

Figure 4: Hybrid Class-Database Diagram
Much like the system design, the project’s database structure went through a variety of minor changes throughout the development process. As the design of the system was configured to better suit the requirements of the sponsor and as new requirements came about, these changes needed to be reflected in the database schema as well. Fortunately, these changes never caused major issues with development and came about naturally as development progressed. As individual features were implemented, occasionally oversights became apparent in the current database design. The most common were missing fields that were necessary, a primary/foreign key that was not placed, or even times something as minor as renaming fields for readability’s sake. Ultimately, the database held fairly closely to its initial conception and was effective during development.

**User Interface / Page Flow Design**

After the team had design of the model and created a database design, a basic screen flow was identified. By laying out the flow of pages as presented to the different types of users, the team was able to better convey the actual process of using the application to the sponsor. Additionally, as the page flow was developed page-specific content, such as fields and buttons, were described in a detailed page-flow document. Aside from the benefits of having a concrete representation for the team’s development, these documents were provided to the sponsor’s co-ops to understand what sort of styles would be needed. Figure 5 displays the screen flow diagram.
The screen flow diagram proved to be useful for developing a sense of a natural progression for a user of the system. Its construction allowed the team to work on how the individual components of the system would interact as well as forwarding between controllers, views, and models.

Figures 6 and 7 are two sequence diagrams the team had put together to outline two of the system’s important features. With the help of the screen flow, system design, and
database design the team was able to structure the development in a way that facilitated quick and efficient creation of important features and their interlinking connections.

Figure 6: User enters system and creates a profile

Figure 7: User performs a quick search
Process and Product Metrics
To ensure a successful implementation and delivery of the Co-op Housing Portal, the team selected several process and product metrics to monitor the project. Each individual member was responsible for tracking time and effort. This allowed the team to understand what areas of the project needed the most time assigned and which team members needed to put more effort into the project. These metrics were consolidated and tracked for the entire team. Figure 8 displays a summary of the team’s estimated time versus actual time spent on project tasks during the winter quarter.

![Winter Quarter Estimated Time vs. Actual Time](image)

**Figure 8: Winter Quarter Estimated Time vs. Actual Time**

Figure 8 shows that either the team tended to overestimate the time needed to complete weekly tasks or not all planned tasks were completed. All team members took this analysis into account while estimating time and effort during the spring quarter. Figure 9 shows a summary of the same metric during the spring quarter.
The team continued to overestimate time needed to complete project tasks during the spring quarter. However, the actual time devoted to the project is more consistent across the spring weeks than it was during the winter weeks. Also, it is clear that more time was spent during the spring quarter as each week’s actual time hovers around seventy hours while each week’s actual time during the winter was sporadic from thirty to eighty hours.

The team also tracked project progress by creating a milestone slippage chart. This chart allowed the team to assess where the project was falling behind. The metric was important in that slippages could easily be recognized and therefore reacted to quickly. Figure 10 shows a milestone slippage chart of the most important milestones.
The project started off meeting all milestone deadlines. However, the gathering of the requirements took much longer than expected, causing the SRS to be completed at a much later time than the due date. This also caused the design document due date to slip. As the project continued, the milestones were completed almost on time, give or take a couple of days. However, it is worth noting that they were not on time. This metric shows that the team could have done a better job planning the milestones to ensure that they were completed on time.

In addition to tracking time and effort and progress, the team also initially planned to track defects per use case. However, the team later changed the metric to tickets per use case. This new metric allowed the team to view which use cases requires the most work and therefore the most time. The tickets were also broken up into the following three categories: task, defect, and enhancement. This further allowed the team to decide the priorities of tickets.

Figure 11 displays the tickets per use case metric related to normal user features. Figure 12 displays the tickets per use case metric related to housing provider and administrator features. The Use Case # references the uses cases listed in the SRS document. In the cases of Use Case 0 and Use Case 30, these do not refer to use cases in the SRS. Use Case 0 describes all tickets related to the database and logging into the system. Use Case 30 describes all administrator tickets that do not fall into another category.
The team classified tasks as adding new features to the system. Defects were classified as actual errors in the system. Enhancements were classified as changes that help the system to be more useful and efficient.

After classifying the three ticket types, the team could analyze the tickets per use case metric. As expected, the normal user features contained the majority of the tickets. Since there were more requirements for normal user features, it followed that there would be more tickets. Also, it is easy to see that the majority of tickets for normal user features were either tasks or enhancements. This shows that the team’s implementation worked as expected as there was a low defect count. The housing provider and administrator feature tickets also follow this analysis.

The creation and tracking of tickets greatly helped the team to keep organized. It also displayed a clear definition of exactly what needed to be completed.

**Product State at Time of Delivery**

Using the SRS, a detailed list of formalized requirements, the team was able to facilitate the development in such a way as to have the system deliverable by the given due date. The project is feature complete in accordance to the SRS document with the exception of a searching web service API, which was set as a lower priority than some of the more vital requirements needed for the system. This means that the higher priority features were completed before those of lower priority. The failure to implement the web service searching API was simply an issue of time.
An additional issue with the system that was never implemented fully was the styling of the pages. Although not necessarily a requirement laid out in the SRS, the styling of the pages was discussed amongst the team and the sponsor. The designer in the co-op office would provide some styling to the HTML of the site. This was meant to be implemented by week seven of the spring quarter for usability testing, but also due to time constraints it was not fully implemented. To mitigate this issue early on, the team decided to use a Model-View-Controller design to separate out the views from the business logic of the system. This way the designer had the ability to edit the HTML of the pages without having to worry about breaking any code.

As far as feature creep, the system stayed fairly close to the SRS document. Some minor changes were done to features. For example, the ability to block a profile or house in the system was separated out into two different types of blocks, a user block and an admin block. This allowed the system to differentiate from a profile/house being blocked for offensive reasons by an admin and those being blocked by the end user because they are no longer available. In the end the delivered product was derived directly from the SRS document and fulfilled all the client’s expectations.

**Project Reflection**

Overall the project went well. The team spent a little more time with requirement elicitation than expected or would have liked. However, because the requirements were so well laid out, the team was allowed to minimize the amount of new features added to the system as development went along. In retrospect, it would have been better to have chosen an agile process to get development started a little earlier. Also, it would have been wise to have developed prototypes to help drive the requirements gathering process.

There were also some issues with getting the development environments setup, both locally and remotely. A lot of time was spent on getting the code repository to be pushed to staging and configuring Apache to get it to all work correctly.

On a positive note, communication with the sponsor went really well. The team was able to meet on a weekly basis with the sponsor and also through email when necessary.

The technologies the team was able to use were restricted to those that RIT was able to provide and maintain, which brought about a learning curve for the team. The team was able to minimize the impact of the learning curve of new technologies by using independent time to study each of the unfamiliar technologies being used. One technology that brought about some issues to the team was the CakePHP framework. Although it provided a good basis for the Model-View-Controller, it made it difficult to implement some features like public and private paging behind LDAP. This was because of how routing was setup in the CakePHP framework. In retrospect, looking at some other PHP frameworks, like the Zend framework, might have led to less of a hindrance in certain development aspects.

Overall, the project was a good learning experience. It gave the team some experience in not only developing a web application, but also working through the whole development
cycle of a software product. The weekly meetings with the client, for example, started out a little unorganized, but as the meetings went on the team became more comfortable with the client-developer interaction and the productivity began to increase.

References
Trac: http://trac.edgewall.org/
CakePHP: http://cakephp.org/